

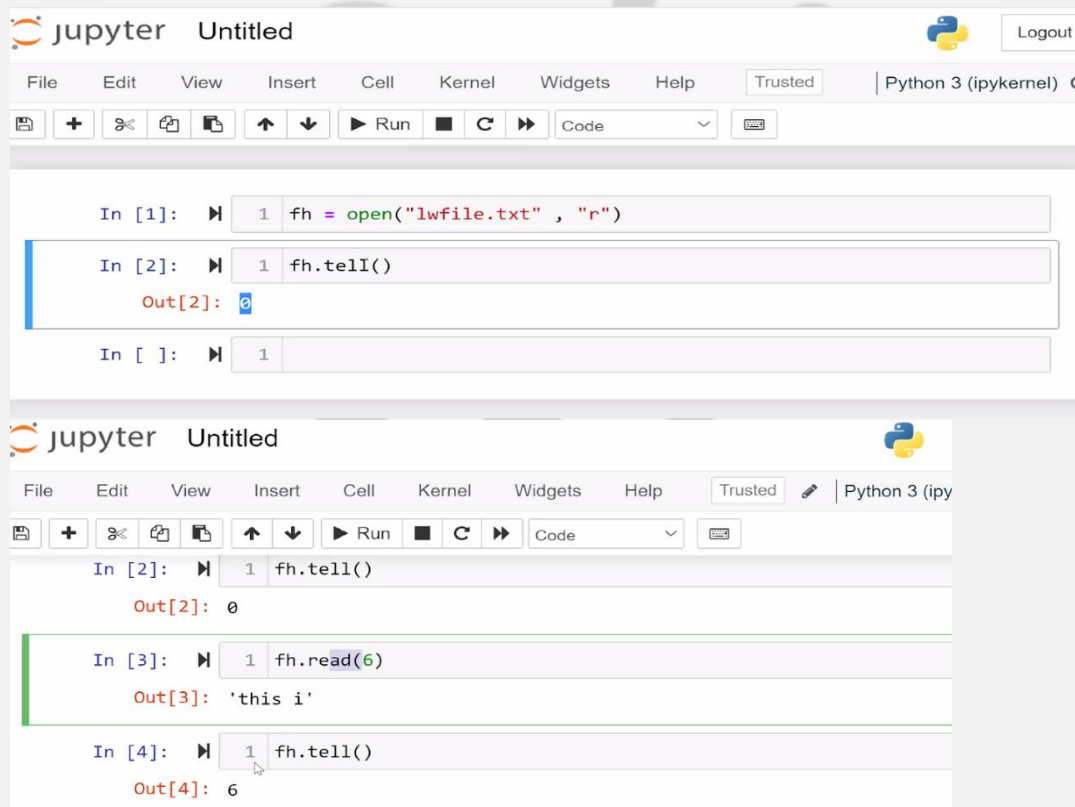


## **Python Session Summary 19-06-2023**

### **File handling**

- File handling allows you to perform various operations on files, including writing, reading, and appending content.
- To read the content of a file, you can open the file in read mode ("r"), use the read() method to retrieve the content, and to write content to a file, you can open the file in write mode ("w") or append mode ("a") using the open() function.
- **Reading a file**
  - To open a file we use the open() function
  - The "r" mode is used to open the file in read-only mode, allowing you to read the contents of the file but not modify or delete anything inside.

- When you open a file, Python creates a file object, which acts as a file handler. This file object allows you to perform various operations on the file, such as reading, writing, or closing the file.
- To determine the current position within the file, you can use the `tell()` method. The `tell()` method returns the current file position. This can be helpful if you need to keep track of your position while reading or writing the file.
- Here's an example of how you can open a file in read-only mode, read its contents, and use the `tell()` method to get the current file position:



```
In [1]: 1 fh = open("lwfile.txt" , "r")

In [2]: 1 fh.tell()
Out[2]: 0

In [ ]: 1
```

```
In [2]: 1 fh.tell()
Out[2]: 0

In [3]: 1 fh.read(6)
Out[3]: 'this i'

In [4]: 1 fh.tell()
Out[4]: 6
```

- **Seek() function**

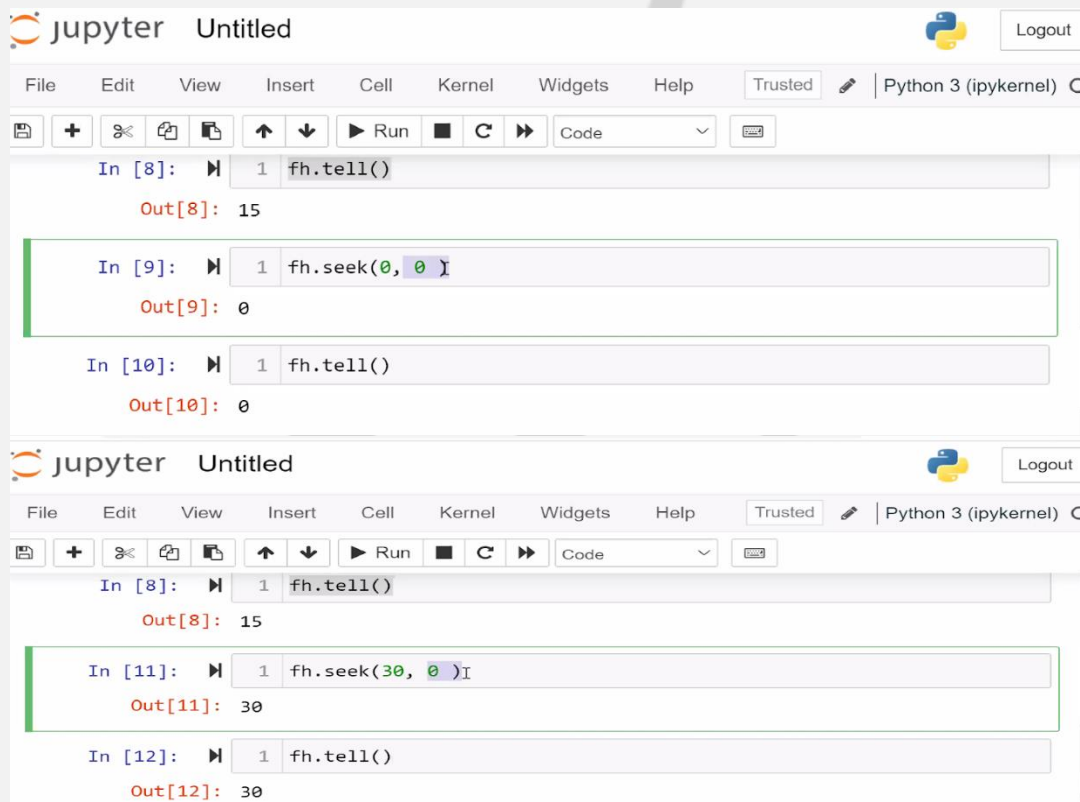
- The `seek()` function allows you to move the cursor to a specific position in the file, which can be useful for random access or when you want to read data from a specific location.

- The `seek()` function takes two arguments: `offset` and `whence`. The `offset` specifies the number of bytes to move the file position, and the `whence` parameter determines the reference point from where the offset is applied. Here are the three possible values for `whence`:

0 (default): The offset is relative to the beginning of the file.

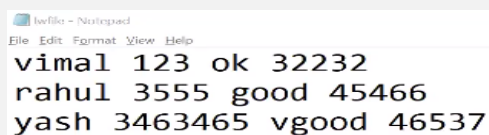
1: The offset is relative to the current file position.

2: The offset is relative to the end of the file.



The image shows two screenshots of a Jupyter Notebook interface. The top screenshot shows a sequence of three code cells: 1. `In [8]: fh.tell()` with output `Out[8]: 15`. 2. `In [9]: fh.seek(0, 0)` with output `Out[9]: 0`. 3. `In [10]: fh.tell()` with output `Out[10]: 0`. The bottom screenshot shows another sequence of three code cells: 1. `In [8]: fh.tell()` with output `Out[8]: 15`. 2. `In [11]: fh.seek(30, 0)` with output `Out[11]: 30`. 3. `In [12]: fh.tell()` with output `Out[12]: 30`. Both screenshots show the Jupyter Notebook menu bar and toolbar.

- In Python, when working with text, characters are indeed treated individually. You can iterate over a string or a file line by line, character by character, or word by word, depending on your specific requirements.



The image shows a Notepad window with the following text:

```
vimal 123 ok 32232
rahul 3555 good 45466
yash 3463465 vgood 46537
```

```
In [29]: 1 myfile = open("lwfile.txt") I
In [30]: 1 myfile.read()
Out[30]: 'vimal 123 ok 32232\nrahu1 3555 good 45466\nnyash 3463465 vgood 46537\n\n'
In [28]: 1 myfile.close()
```

- If you want to traverse a file by line, you can iterate over the file object using a for loop.

```
In [34]: 1 myfile = open("lwfile.txt")
In [ ]: 1 for x in myfile:
        2     print(x) I
```

- In Linux, the grep command is commonly used to search for specific words or patterns in files.



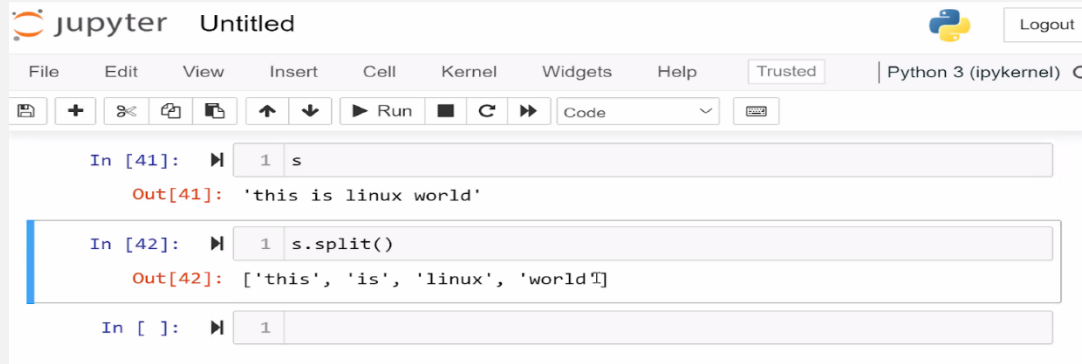
The screenshot shows a terminal window with the following commands and output:

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/summertraining2023
$ cat lwfile.txt
vimal 123 ok 32232
rahu1 3555 good 45466
yash 3463465 vgood 46537

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/summertraining2023
$ grep rahu1 lwfile.txt
rahu1 3555 good 45466

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/summertraining2023
$
```

- If you want to replicate the behavior of the grep command in Python and search for a specific word in a string or a file, you can use the split() function to split the string into individual words and then check if the desired word is present.

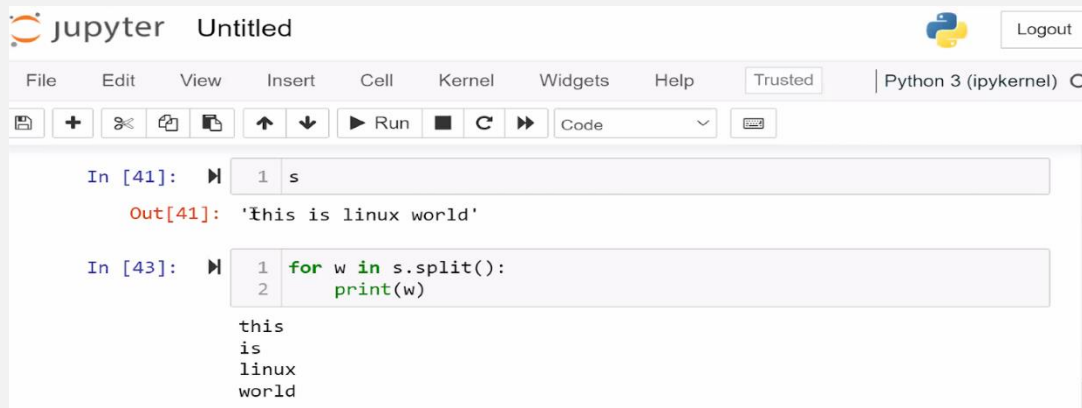


A Jupyter Notebook interface titled 'Untitled' with a Python 3 (ipykernel) environment. The notebook shows three code cells. The first cell contains `s` and outputs `'this is linux world'`. The second cell contains `s.split()` and outputs `['this', 'is', 'linux', 'world']`. The third cell contains `1` and is currently selected.

```
In [41]: 1 s
Out[41]: 'this is linux world'

In [42]: 1 s.split()
Out[42]: ['this', 'is', 'linux', 'world']

In [ ]: 1
```

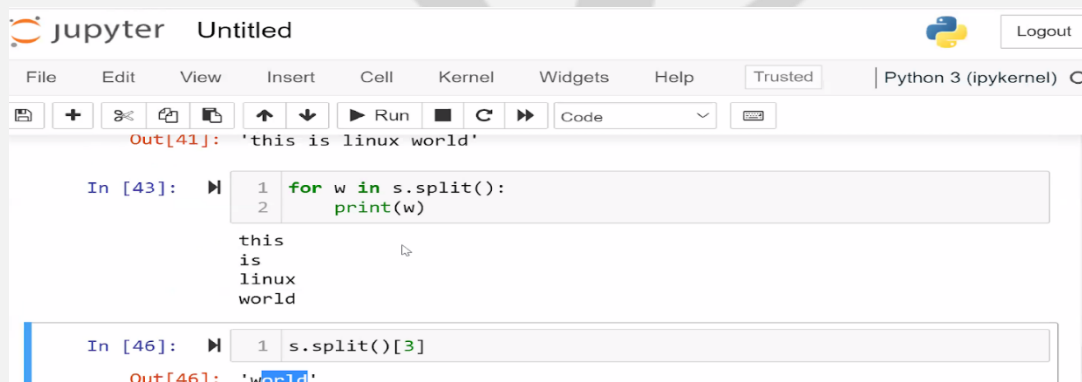


A Jupyter Notebook interface titled 'Untitled' with a Python 3 (ipykernel) environment. The notebook shows three code cells. The first cell contains `s` and outputs `'this is linux world'`. The second cell contains a loop that iterates over `s.split()` and prints each word, resulting in the output `this`, `is`, `linux`, and `world` on separate lines. The third cell is currently selected.

```
In [41]: 1 s
Out[41]: 'this is linux world'

In [43]: 1 for w in s.split():
         2     print(w)
this
is
linux
world
```

- Words can also be extracted by position.



A Jupyter Notebook interface titled 'Untitled' with a Python 3 (ipykernel) environment. The notebook shows three code cells. The first cell contains `s` and outputs `'this is linux world'`. The second cell contains a loop that iterates over `s.split()` and prints each word, resulting in the output `this`, `is`, `linux`, and `world` on separate lines. The third cell contains `s.split()[3]` and outputs `'world'`.

```
In [41]: 1 s
Out[41]: 'this is linux world'

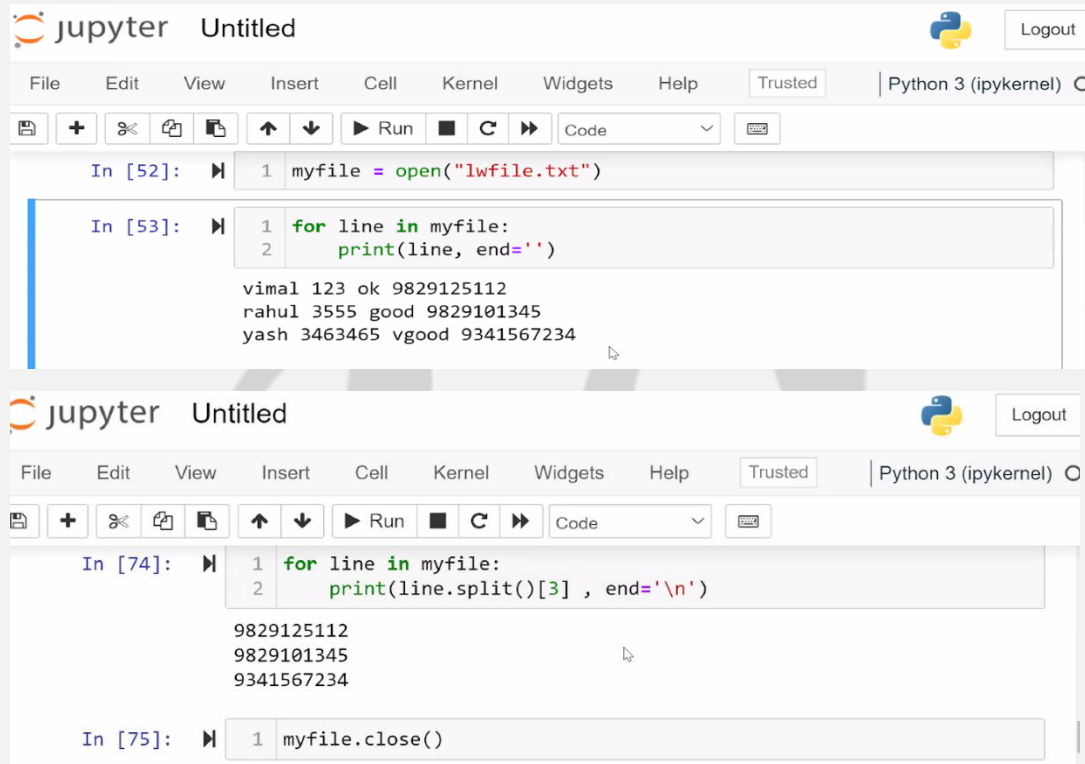
In [43]: 1 for w in s.split():
         2     print(w)
this
is
linux
world

In [46]: 1 s.split()[3]
Out[46]: 'world'
```

- By default, the `print()` function in Python adds an end-of-line character (`'\n'`) at the end of the printed output.
- If you want to change the default end-of-line behavior, you can specify the `end` parameter of the `print()` function. By default, `end='\n'`, but you can

change it to any other character or string. For example, to print without a newline character, you can use:

```
# print("Hello", end="")
```



The first screenshot shows a Jupyter Notebook with the following code and output:

```
In [52]: 1 myfile = open("lwfile.txt")
```

```
In [53]: 1 for line in myfile:
2         print(line, end='')
```

Output:

```
vimal 123 ok 9829125112
rahul 3555 good 9829101345
yash 3463465 vgood 9341567234
```

The second screenshot shows the same Jupyter Notebook with the following code and output:


```
In [74]: 1 for line in myfile:
2         print(line.split()[3] , end='\n')
```

Output:

```
9829125112
9829101345
9341567234
```

```
In [75]: 1 myfile.close()
```

- The `open()` function is used to open the file in write mode ("w"). The file object is assigned to the variable `fh`
- The `write()` function is then used to write the string to the file. This will overwrite any existing content in the file.



The screenshot shows a Jupyter Notebook titled 'Untitled' with a Python 3 (ipykernel) environment. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The code cells are as follows:

```
Out[46]: 'world'
```

```
In [76]: 1 fh = open("mylw123.txt" , mode="w")
```

```
In [78]: 1 fh.write("this is 1st line\nsecline\nnthrid sdbf\n4th bye\n")
```

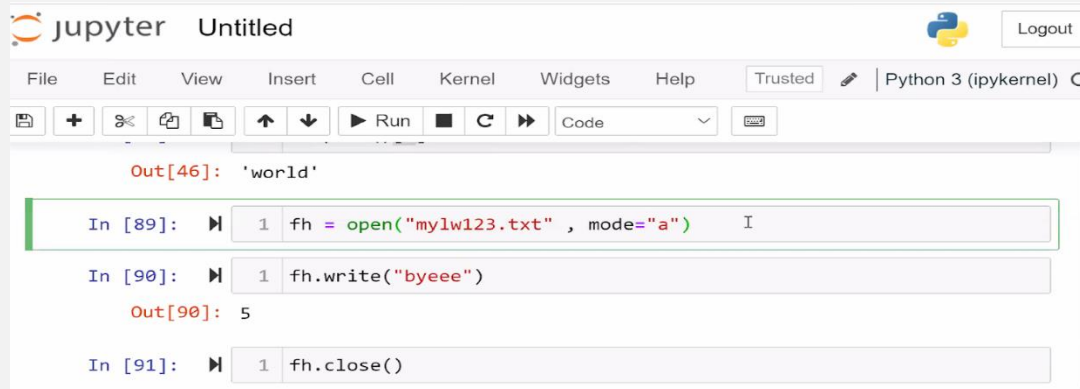
```
Out[78]: 44
```

```
In [ ]: 1 fh.close()
```

- We close the file using the close() function. It's essential to close the file after writing to ensure that all changes are saved and resources are freed.

## Steps to append data to a File in Python

- **Step 1** – Check if the file exists in the specified Path & it has written permissions.
- **Step 2** – Open the file in append mode
  - You first need to open the file in append mode. You can do it with the open() function.
  - When opening the file, you should specify the file name and the mode in which you want to open the file.
  - To open a file in append mode, use the 'a' mode.
- **Step 3** – Append data to file



A screenshot of a Jupyter Notebook interface titled 'Untitled'. The interface includes a top bar with the Jupyter logo, a 'Logout' button, and a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, Help. Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, running, and other actions. The notebook content shows the following code and output:

```
Out[46]: 'world'
```

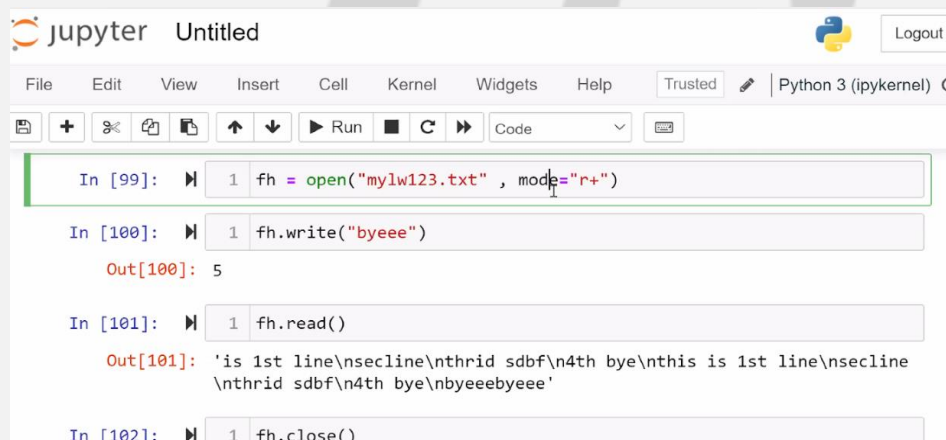
```
In [89]: 1 fh = open("mylw123.txt", mode="a")
```

```
In [90]: 1 fh.write("byeee")
```

```
Out[90]: 5
```

```
In [91]: 1 fh.close()
```

- **Step 4 – Close the file**
- If you want to perform both write and read operations on a file using file handling in Python, you can use the r+ mode when opening the file. The r+ mode allows you to read from and write to the file.



A screenshot of a Jupyter Notebook interface titled 'Untitled'. The interface includes a top bar with the Jupyter logo, a 'Logout' button, and a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, Help. Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, running, and other actions. The notebook content shows the following code and output:

```
In [99]: 1 fh = open("mylw123.txt", mode="r+")
```

```
In [100]: 1 fh.write("byeee")
```

```
Out[100]: 5
```

```
In [101]: 1 fh.read()
```

```
Out[101]: 'is 1st line\nsecline\nnthrid sdbf\n4th bye\nthis is 1st line\nsecline\nnthrid sdbf\n4th bye\nbyeebyeee'
```

```
In [102]: 1 fh.close()
```