COL1000
Introduction to Programming

Priyanka Golia

Most (if not all) of the content is borrowed from Prof. Subodh Kumar's slides

File handing

In any real-world application, data should not vanish when a program ends.
For example:

- A student management system must store student records permanently.
- A text editor saves your work on disk.
- A log file records errors and events over time.

This is where file handling comes in — it allows programs to **store**, **retrieve**, and **update** information on secondary storage (like hard drives).

In Python, files are objects that we can manipulate using built-in functions such as open(), read(), write(), and close().

File Paths: Relative vs Absolute

Absolute Path: The complete path from the **root directory** to the file.
Example (Windows):
C:\Users\Priyanka\Documents\data.txt
Example (Linux/Mac):
/home/priyanka/Documents/data.txt

Path relative to the current working directory.
Example:
If your script is in /home/priyanka/my.py, and the file is in /home/priyanka/data/marks.txt,
then the relative path is: data/marks.txt

```
import os
print(os.getcwd())   # get current working directory
```

```
os.chdir("/path/to/folder")
```

Now, the program will consider you in “folder”.

```
os.path.exists(“filename.txt”)
```

Return True if file is there, else return False.

Files

There are two main categories of files:

- Text files
  Store data in human-readable format (plain text).
  Example: .txt, .csv, .py
- Binary files
  Store data in machine-readable format.
  Example: .jpg, .mp3, .exe

We will primarily focus on **text files** here.

Files

There are two main categories of files:

- Text files
  Store data in human-readable format (plain text).
  Example: .txt, .csv, .py
- Binary files
  Store data in machine-readable format.
  Example: .jpg, .mp3, .exe

We will primarily focus on **text files** here.

Files have persistent names (string, assigned at creation time, but changeable later)

- ➡ One file has only one name
- ➡ Organized into directories — Directories contain files and other directories. File names in one directory are unique. Files in different directory may share name.

Data in file is a sequence of numbers

- ➡ Often represents Unicode or ASCII characters in UTF-8 encoding
- ➡ Can also be 'raw' binary (base-2)

Opening and Closing Files

```
file_object = open(filename, mode)
```

Opening and Closing Files

iterable

```
file_object = open(filename, mode)
```

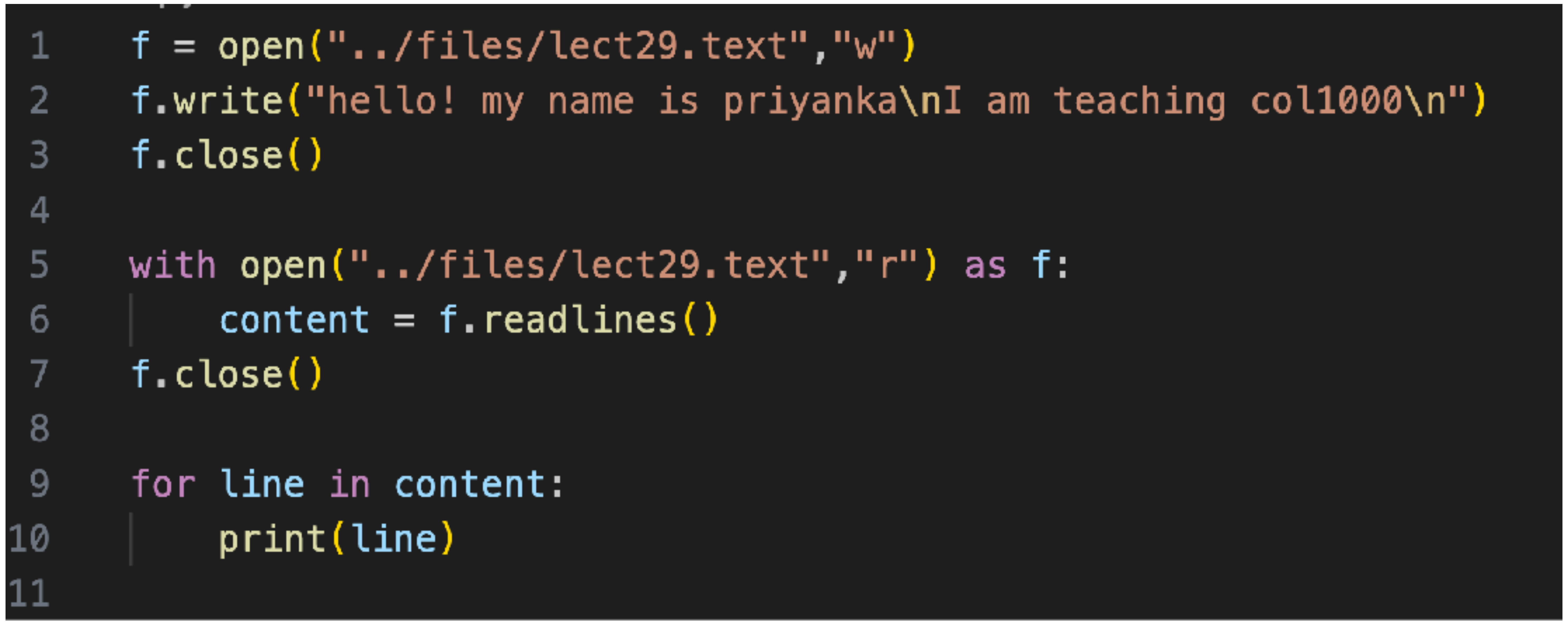
Opening and Closing Files

iterable

```
file_object = open(filename, mode)
```

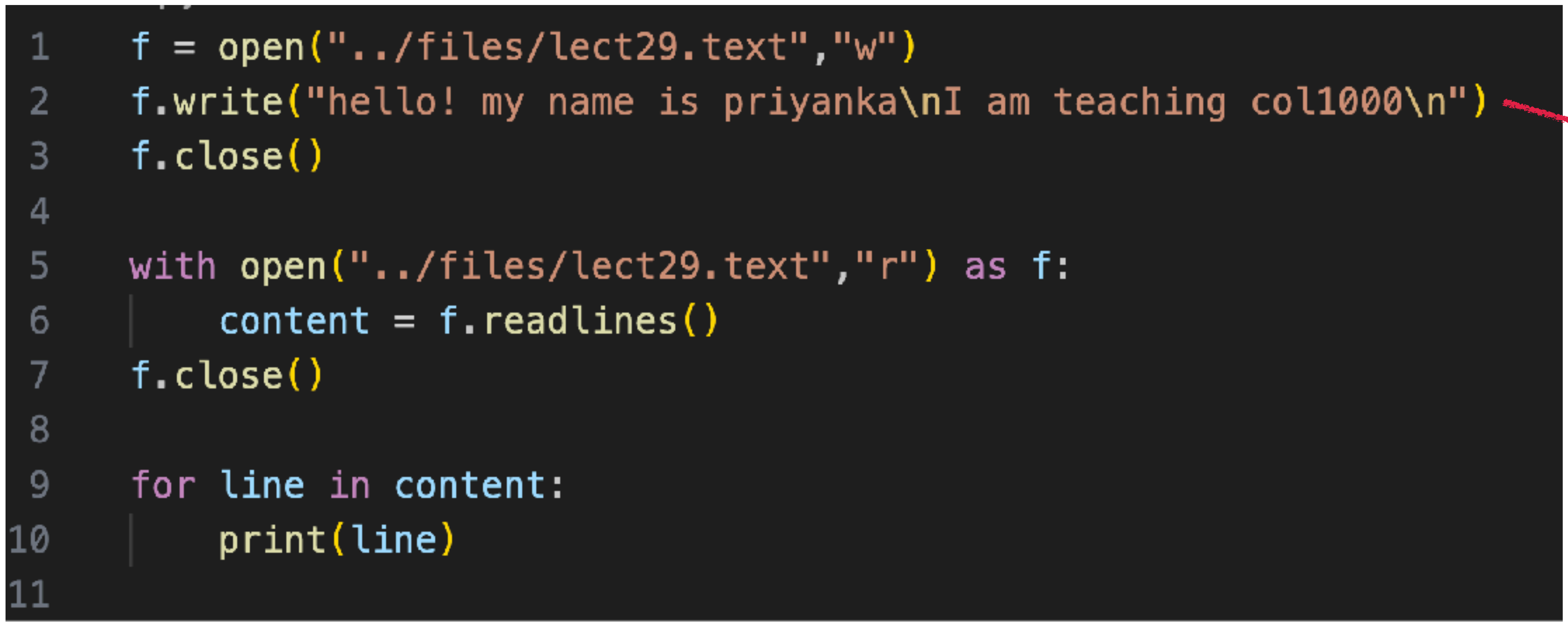
Mode	Meaning	Description
‘r'	Read	Opens file for reading **(default)**. File must exist.
‘w'	Write	**Creates** or **overwrites** a file.
‘a'	Append	Opens for writing at end of file, without truncating.
‘r+'	Read + Write	Opens file for both reading and writing.
‘b'	Binary	Used with above modes, e.g. `'rb'`, `'wb'`.

Opening and Closing Files

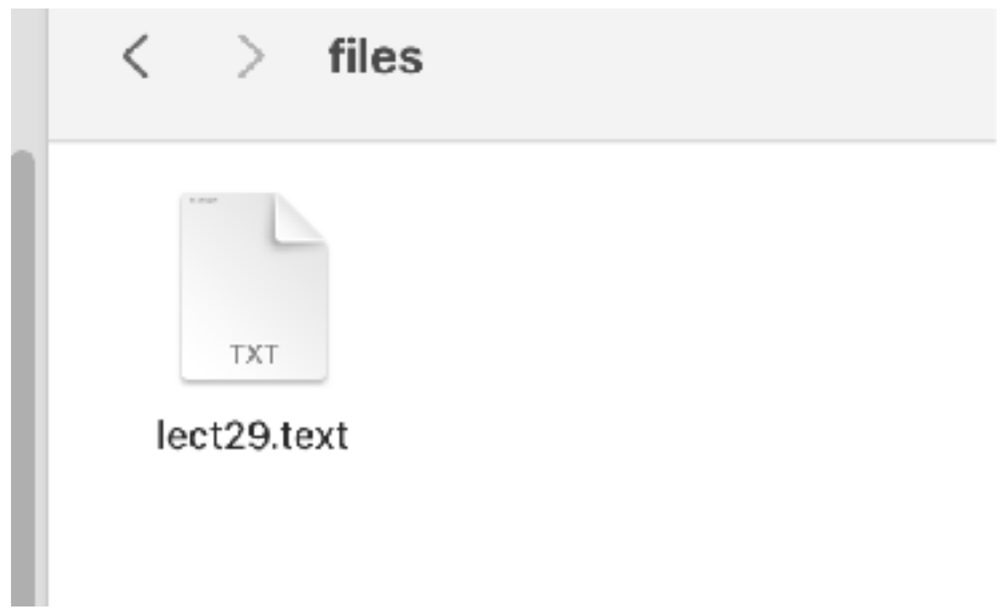


```python
f = open("../files/lect29.text","w")
f.write("hello! my name is priyanka\nI am teaching col1000\n")
f.close()

with open("../files/lect29.text","r") as f:
    content = f.readlines()
f.close()

for line in content:
    print(line)
```

Opening and Closing Files

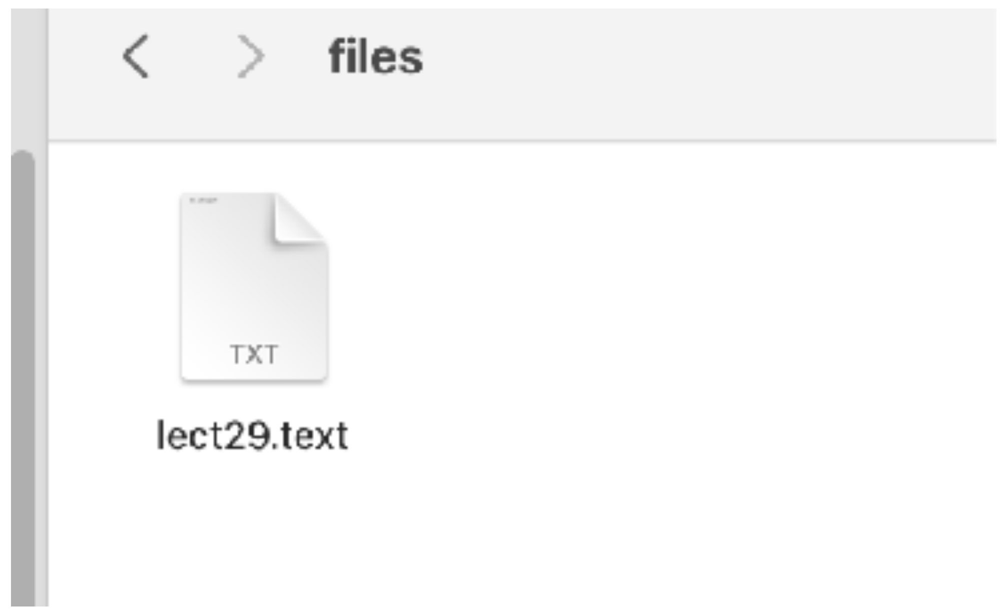


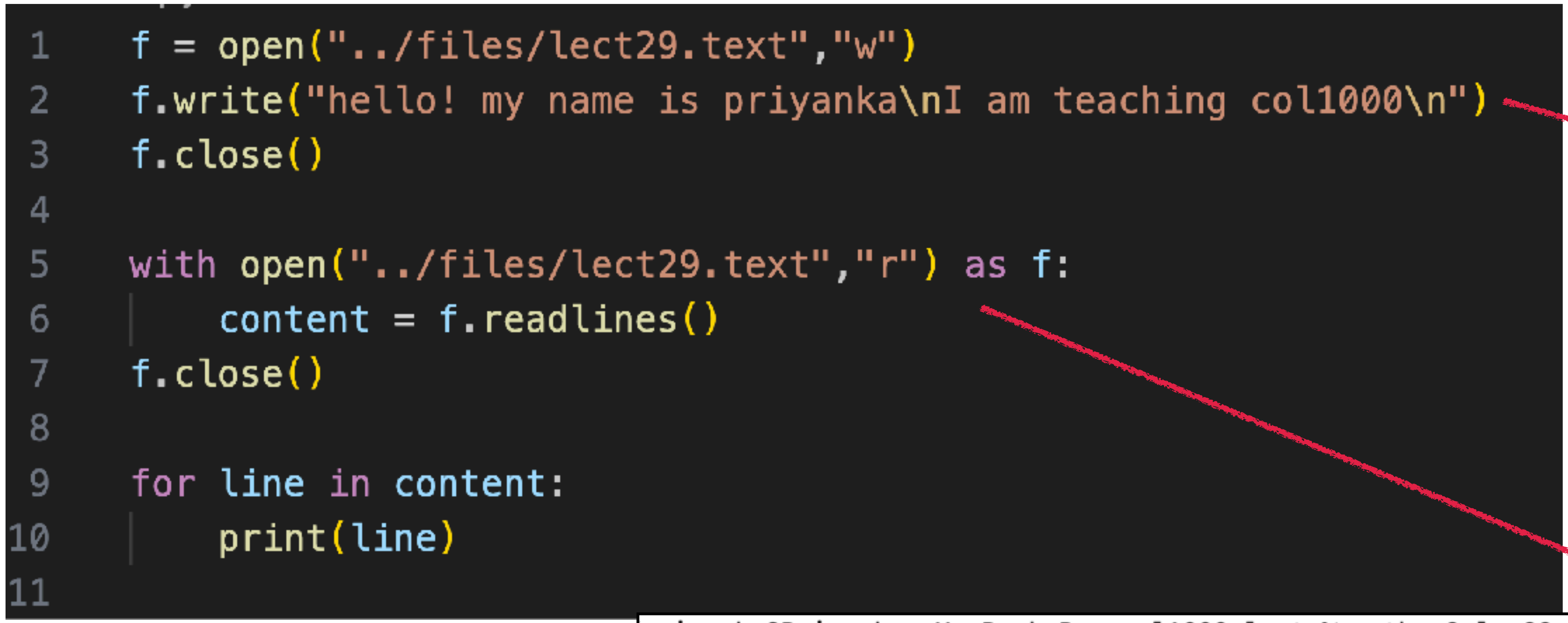
```
f = open("../files/lect29.text","w")
f.write("hello! my name is priyanka\nI am teaching col1000\n")
f.close()

with open("../files/lect29.text","r") as f:
    content = f.readlines()
f.close()

for line in content:
    print(line)

```



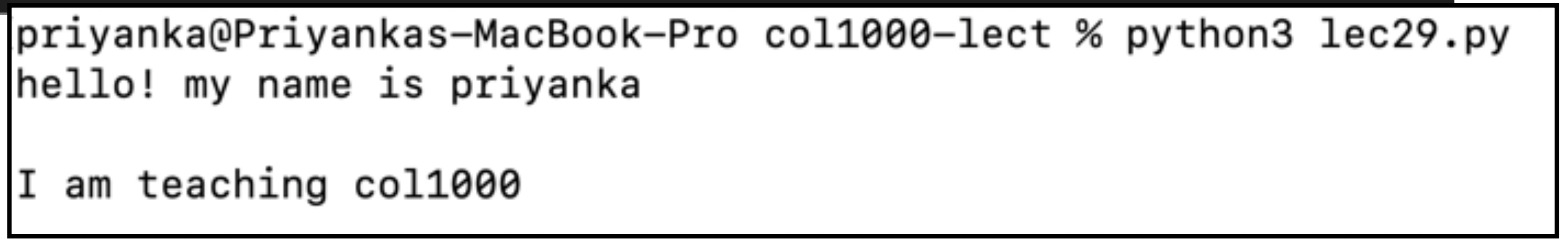
It will create a file named **lect29.txt** in the file directory (one level up from the current directory). If the file does not exist, it will be created. If it already exists, its contents will be erased before writing the new message to it.

Opening and Closing Files





```python
f = open("../files/lect29.text","w")
f.write("hello! my name is priyanka\nI am teaching col1000\n")
f.close()

with open("../files/lect29.text","r") as f:
    content = f.readlines()
f.close()

for line in content:
    print(line)
```

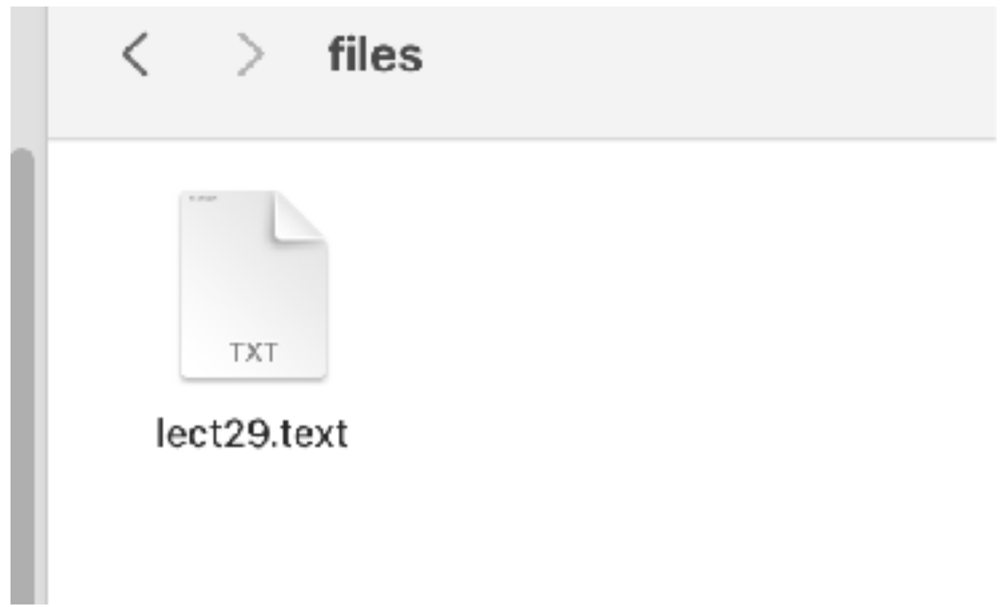


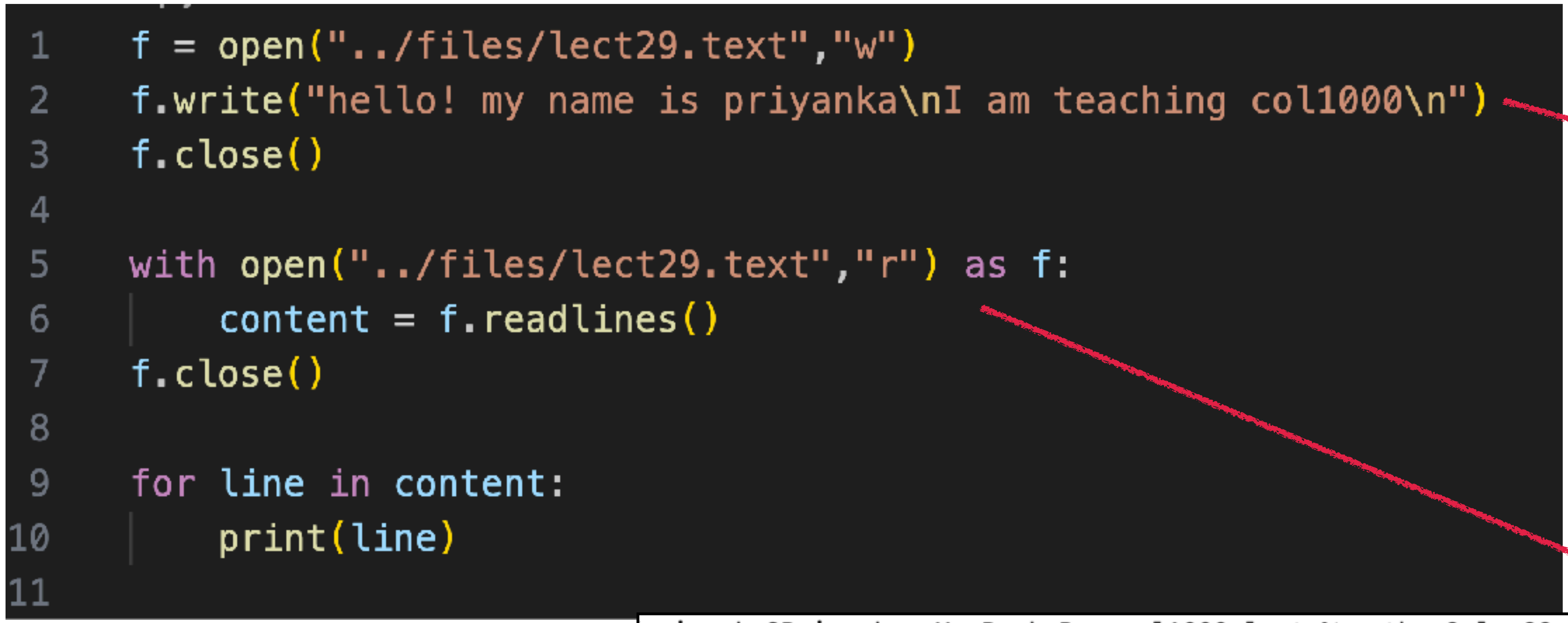
```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
hello! my name is priyanka

I am teaching col1000
```

It will create a file named **lect29.txt** in the file directory (one level up from the current directory). If the file does not exist, it will be created. If it already exists, its contents will be erased before writing the new message to it.

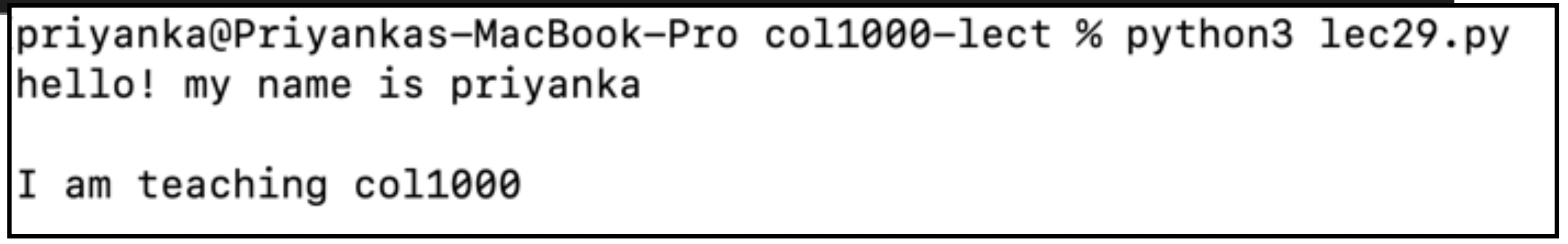
This will open the file in read mode. The contents will be stored line by line. The file object is an iterable, which means you can loop over it to read each line sequentially.

Opening and Closing Files





```python
f = open("../files/lect29.text","w")
f.write("hello! my name is priyanka\nI am teaching col1000\n")
f.close()

with open("../files/lect29.text","r") as f:
    content = f.readlines()
f.close()

for line in content:
    print(line)

```



```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
hello! my name is priyanka

I am teaching col1000
```

It will create a file named **lect29.txt** in the file directory (one level up from the current directory). If the file does not exist, it will be created. If it already exists, its contents will be erased before writing the new message to it.

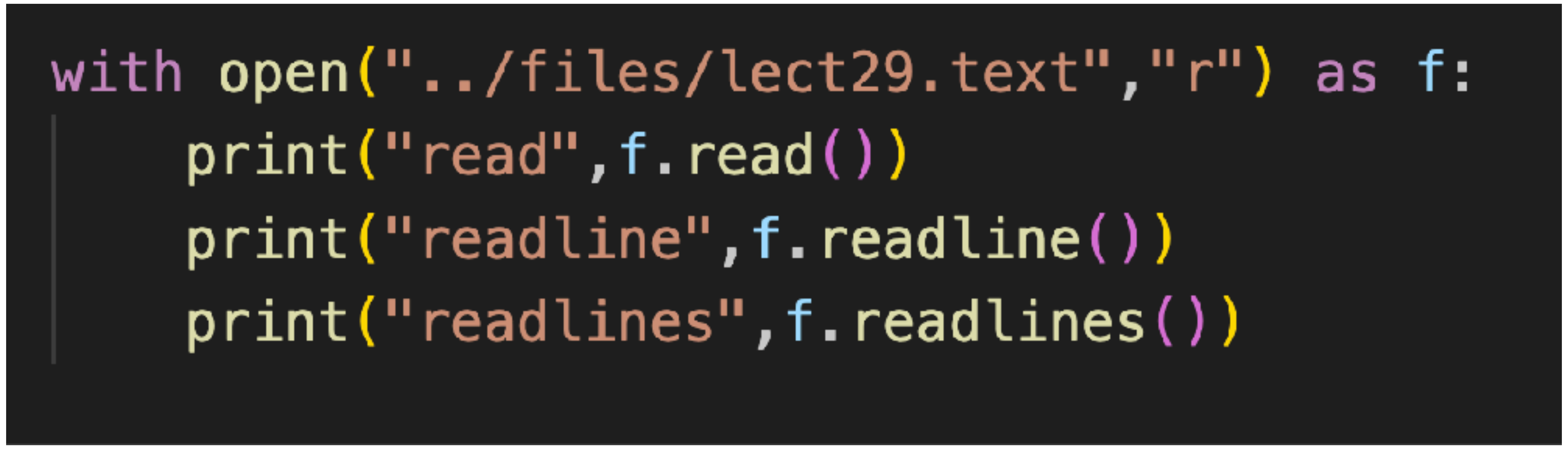
This will open the file in read mode. The contents will be stored line by line. The file object is an iterable, which means you can loop over it to read each line sequentially.

<file_object>.close is important — When you open a file, the operating system allocates resources (like file handles or memory buffers) to it. Closing the file tells the OS that you're done, so these resources can be freed.

Always close — data may not be saved or file remains locked.

The "with" statement in Python automatically closes the file for you.

Reading Files



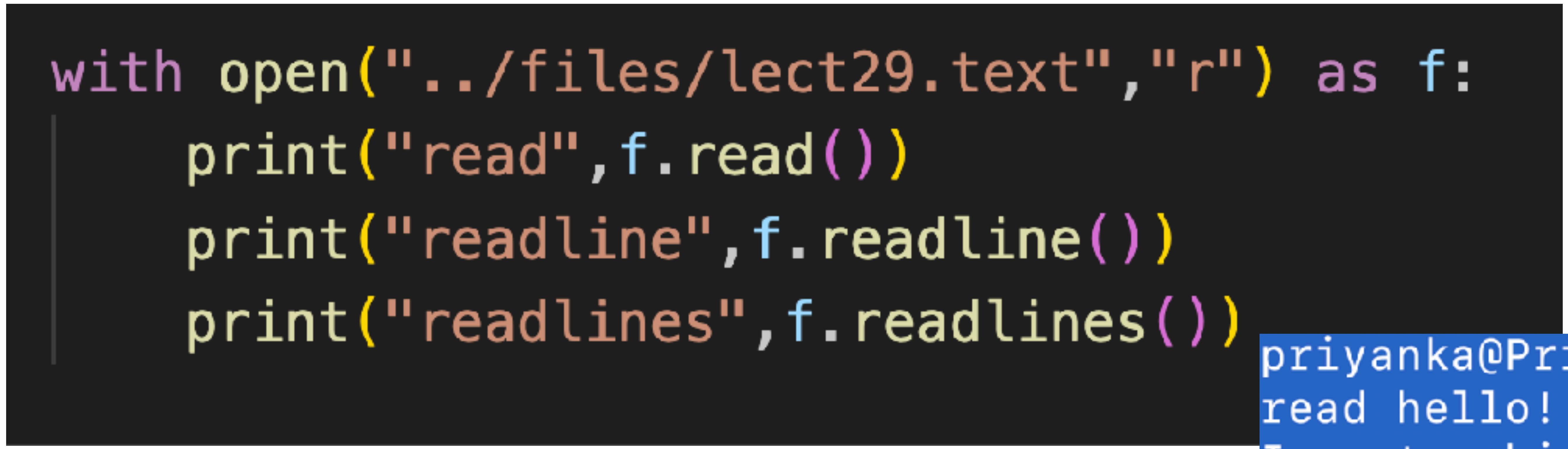
```
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

f.read() — Reads the entire file content at once (including \n newlines). Returns a single string
You can optionally specify how many bytes/characters to read: f.read(10) — reads first 10 characters only.

f.readline() — Reads only one line (up to \n) at a time. Returns a string that includes the newline character at the end. When called again, it continues from where it left off.

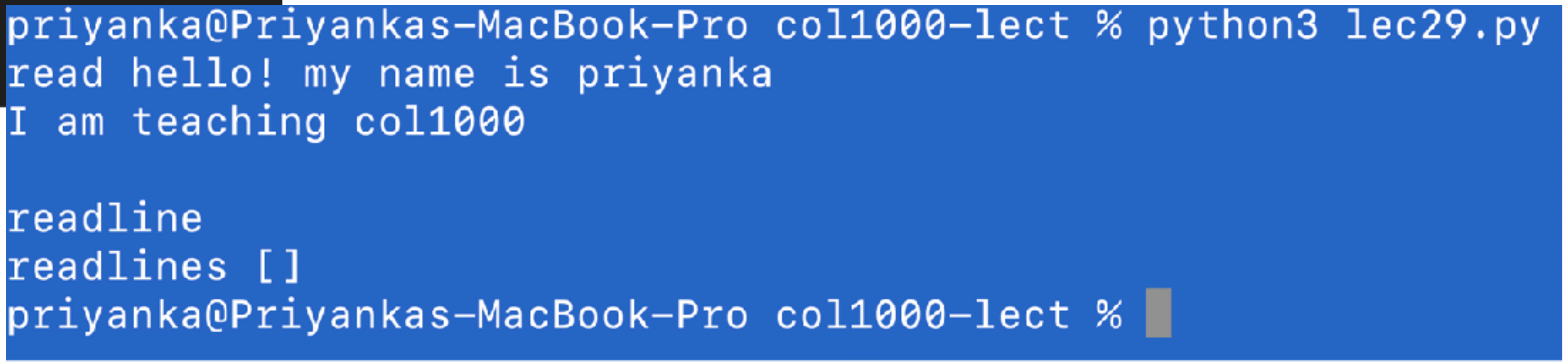
f.readlines() — Reads all lines at once.Returns a list of strings, one per line (each ending with \n).

Reading Files



```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Output



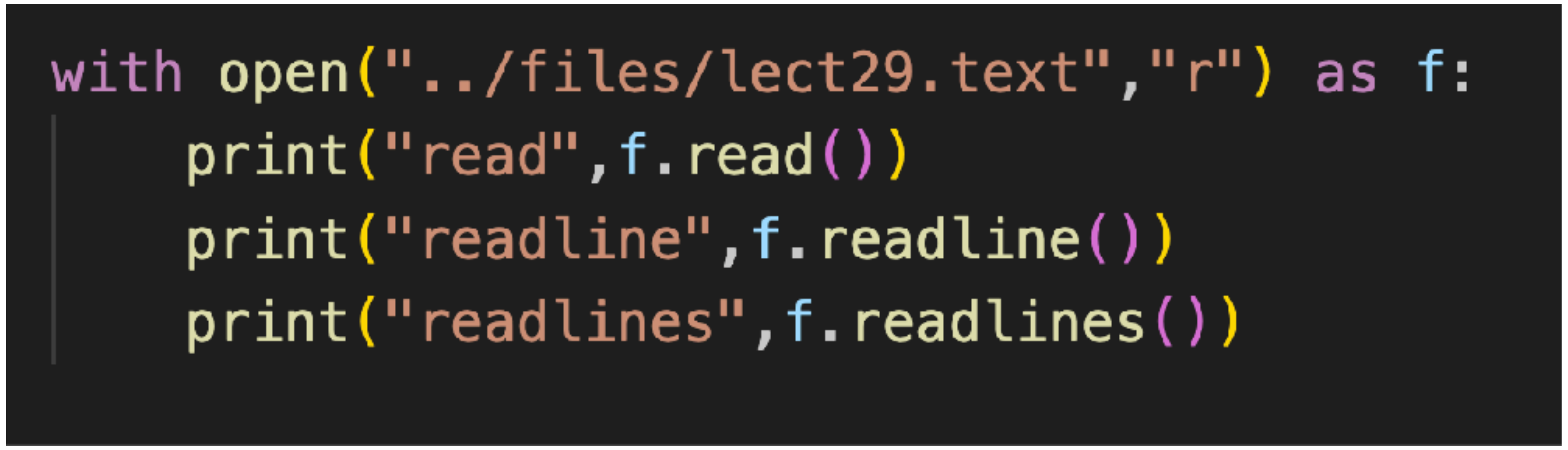
```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
read hello! my name is priyanka
I am teaching col1000

readline
readlines []
priyanka@Priyankas-MacBook-Pro col1000-lect %
```

f.read() — Reads the entire file content at once (including \n newlines). Returns a single string
You can optionally specify how many bytes/characters to read: f.read(10) — reads first 10 characters only.

f.readline() — Reads only one line (up to \n) at a time. Returns a string that includes the newline character at the end. When called again, it continues from where it left off.

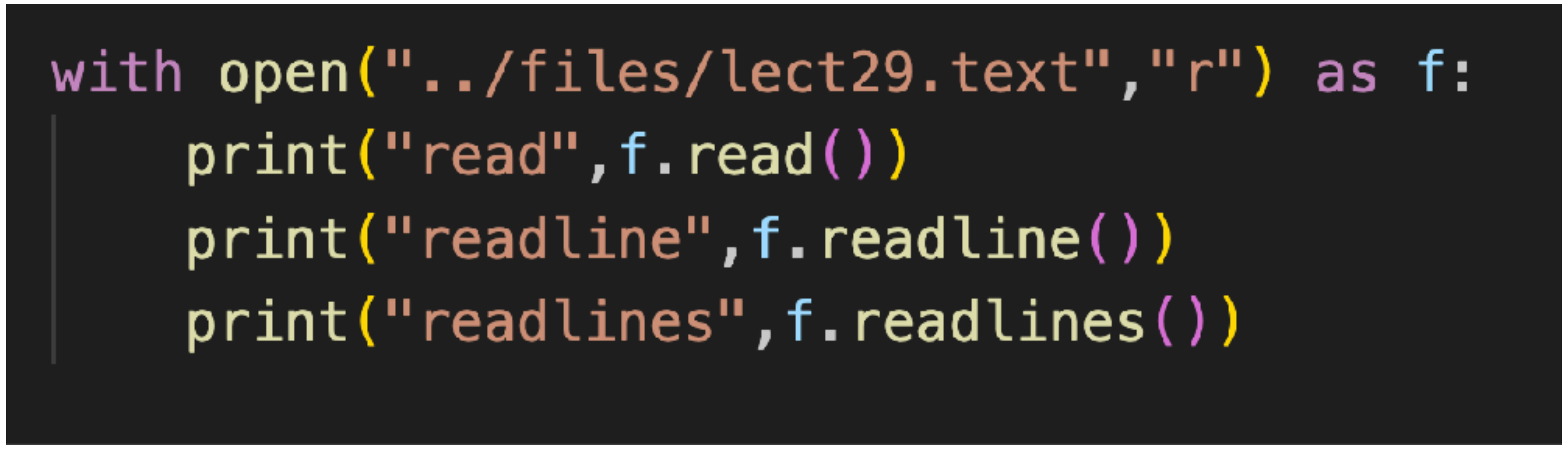
f.readlines() — Reads all lines at once.Returns a list of strings, one per line (each ending with \n).

Reading Files



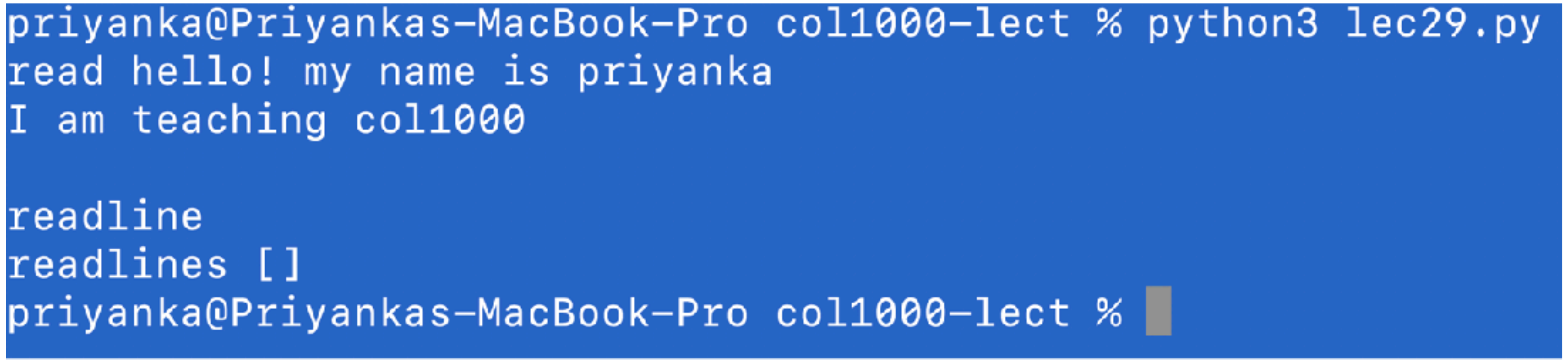
```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Reading Files



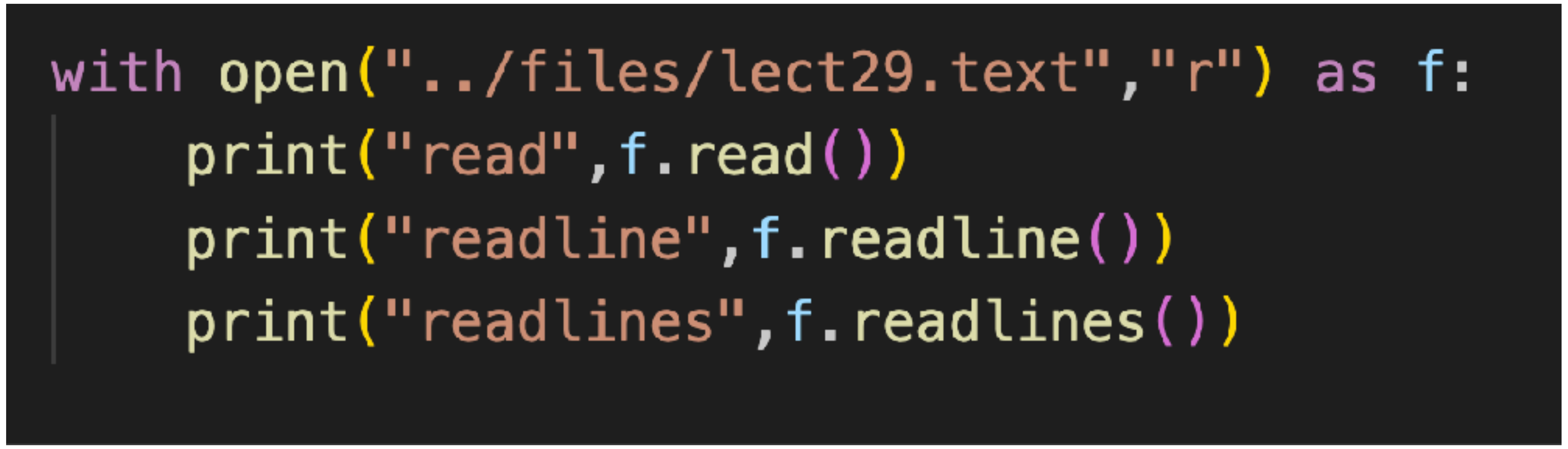
```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Output



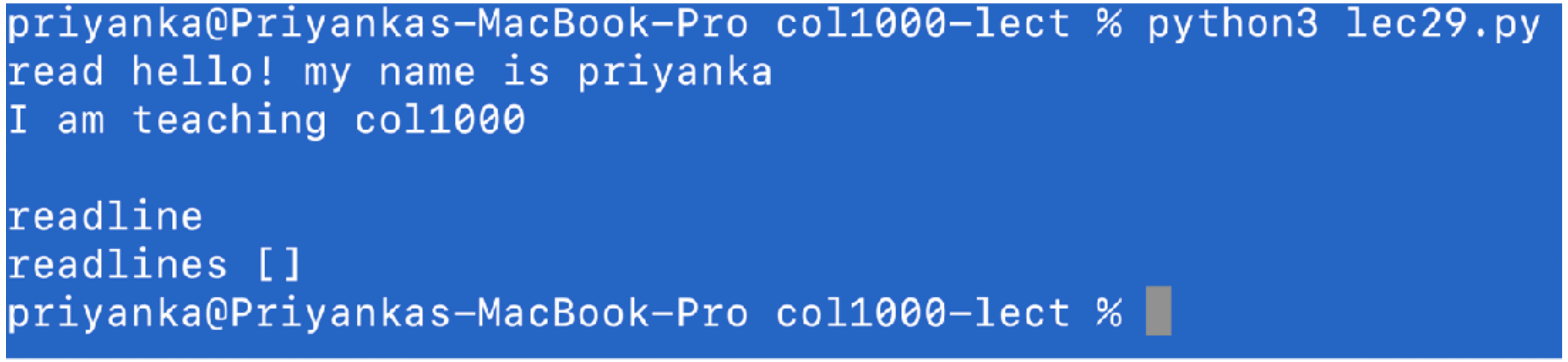
```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
read hello! my name is priyanka
I am teaching col1000

readline
readlines []
priyanka@Priyankas-MacBook-Pro col1000-lect %
```

Reading Files



```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

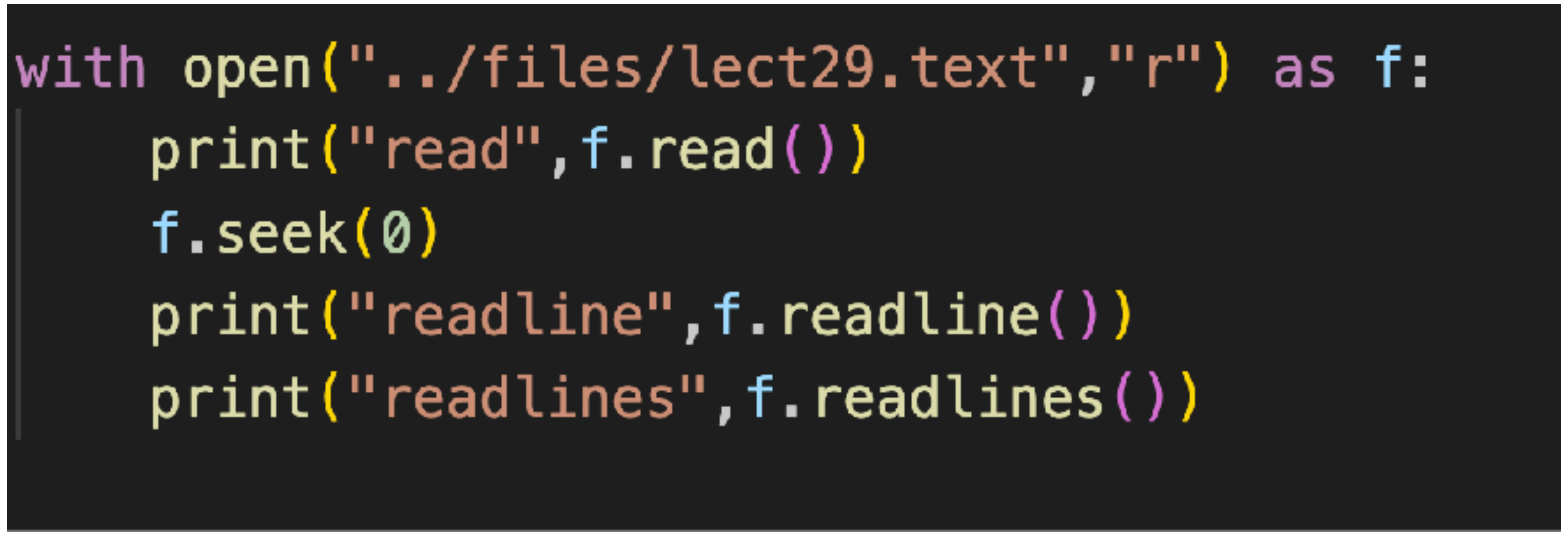
Output



```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
read hello! my name is priyanka
I am teaching col1000

readline
readlines []
priyanka@Priyankas-MacBook-Pro col1000-lect %
```

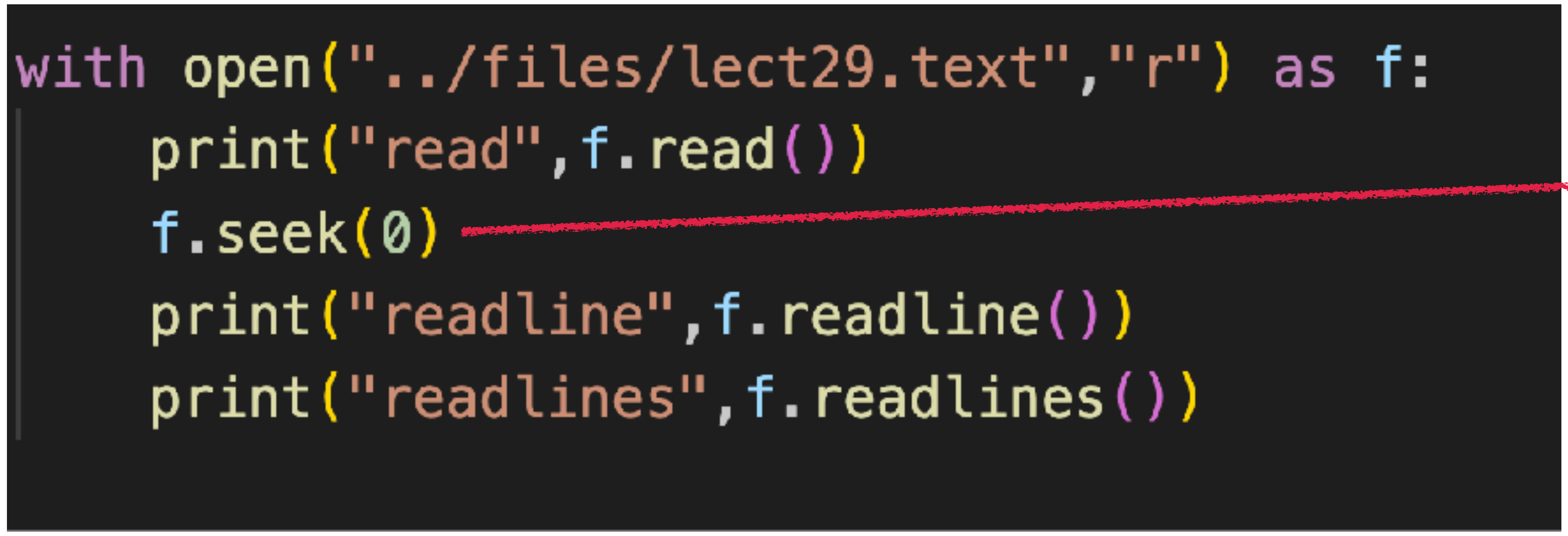
- When you open a file for reading, Python maintains an internal file pointer (like a cursor).
- At the start, it's at position 0 — the beginning of the file.
- As you read data (via .read(), .readline(), or .readlines()), the pointer moves forward.
- If you try to read again, it starts from the current position.
- To go back, you use f.seek(0) — which moves the pointer back to the start.

Reading Files



```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    print("readlines",f.readlines())
```

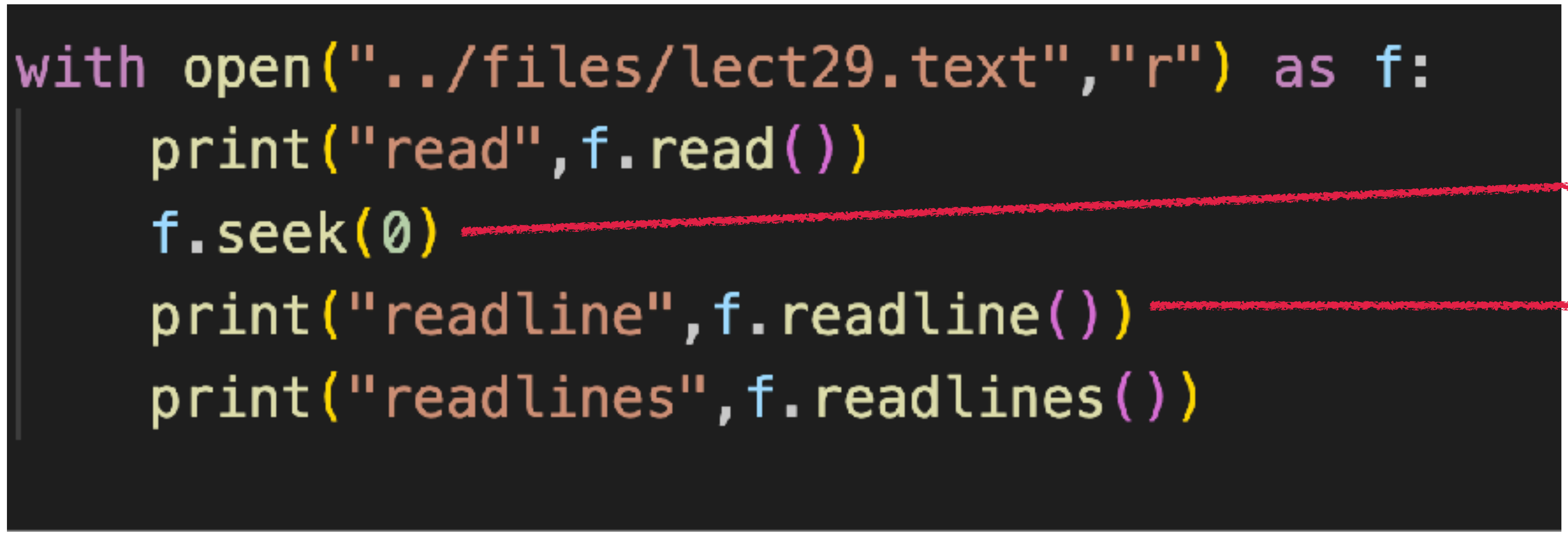
Reading Files



```
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Resetting the pointer to position 0.

Reading Files

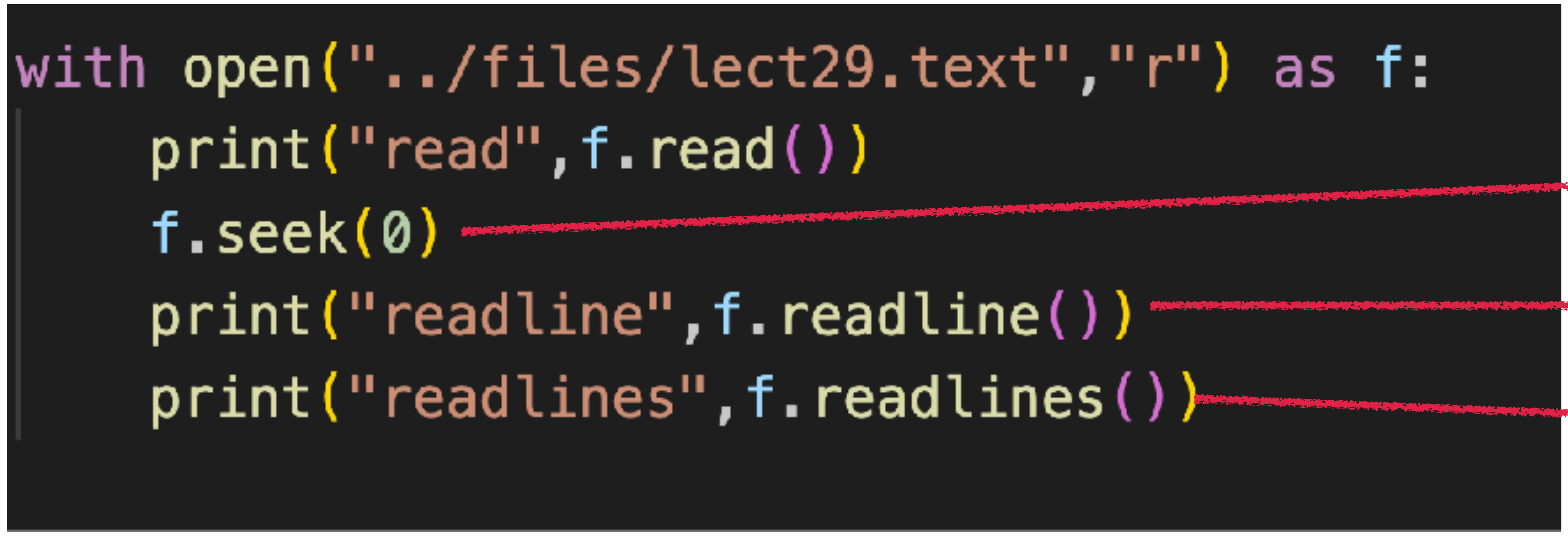


```
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Resetting the pointer to position 0.

Returns the current line (a single line)

Reading Files



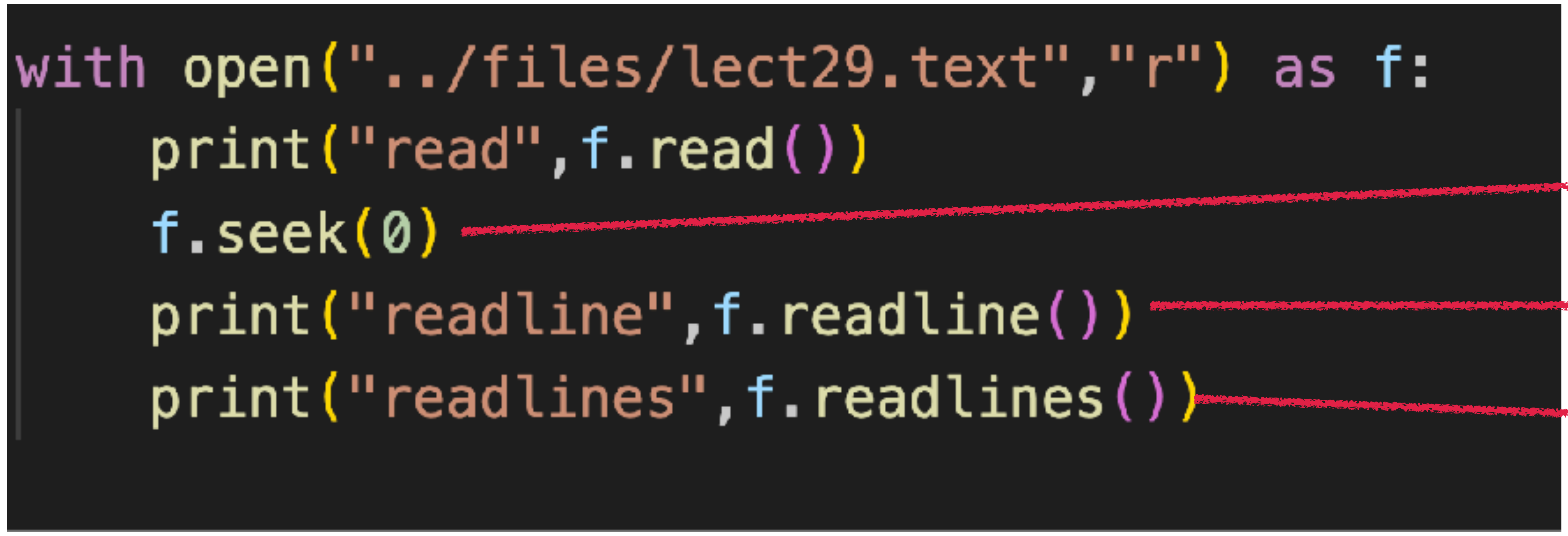
```
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Resetting the pointer to position 0.

Returns the current line (a single line)

Returns the list of lines — from the pointer position.

Reading Files



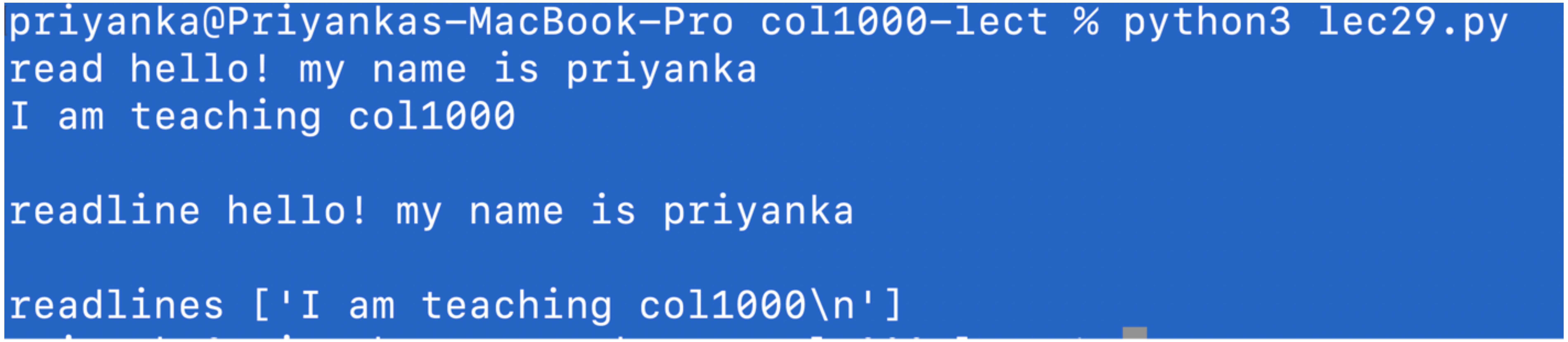
```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Resetting the pointer to position 0.

Returns the current line (a single line)

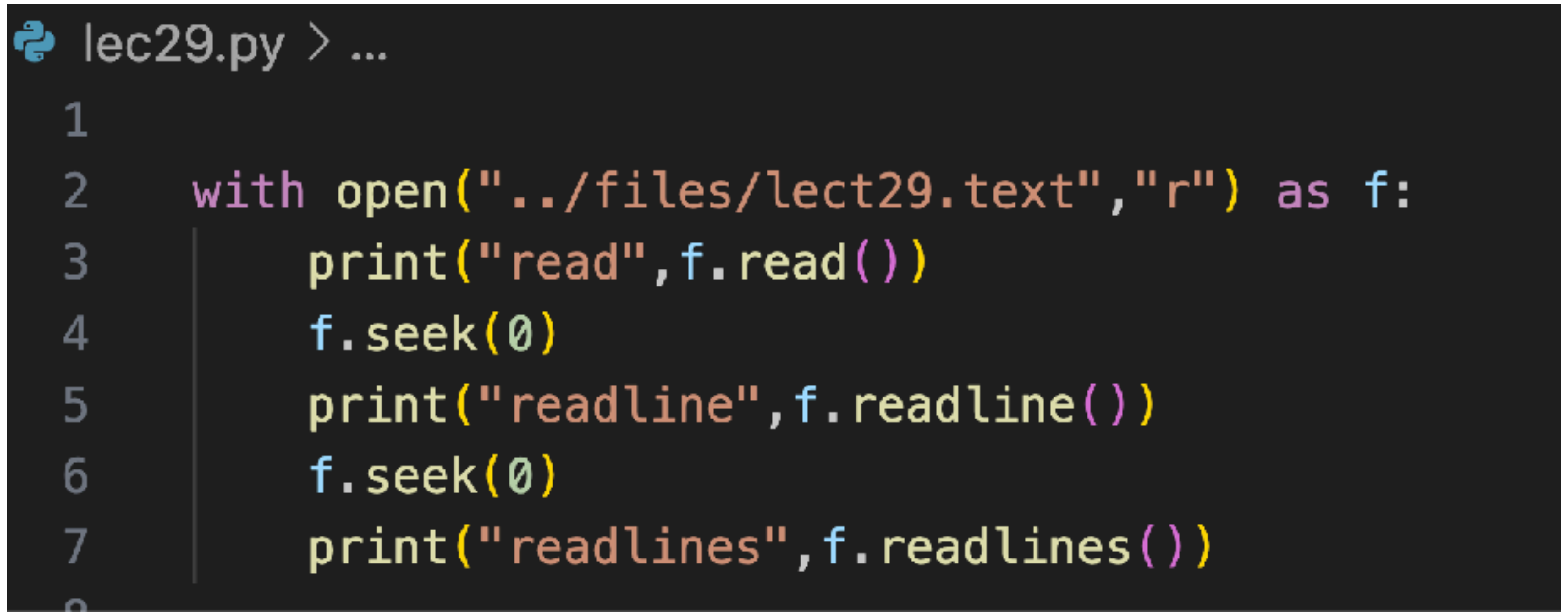
Returns the list of lines — from the pointer position.

Output



```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
read hello! my name is priyanka
I am teaching col1000

readline hello! my name is priyanka

readlines ['I am teaching col1000\n']
```

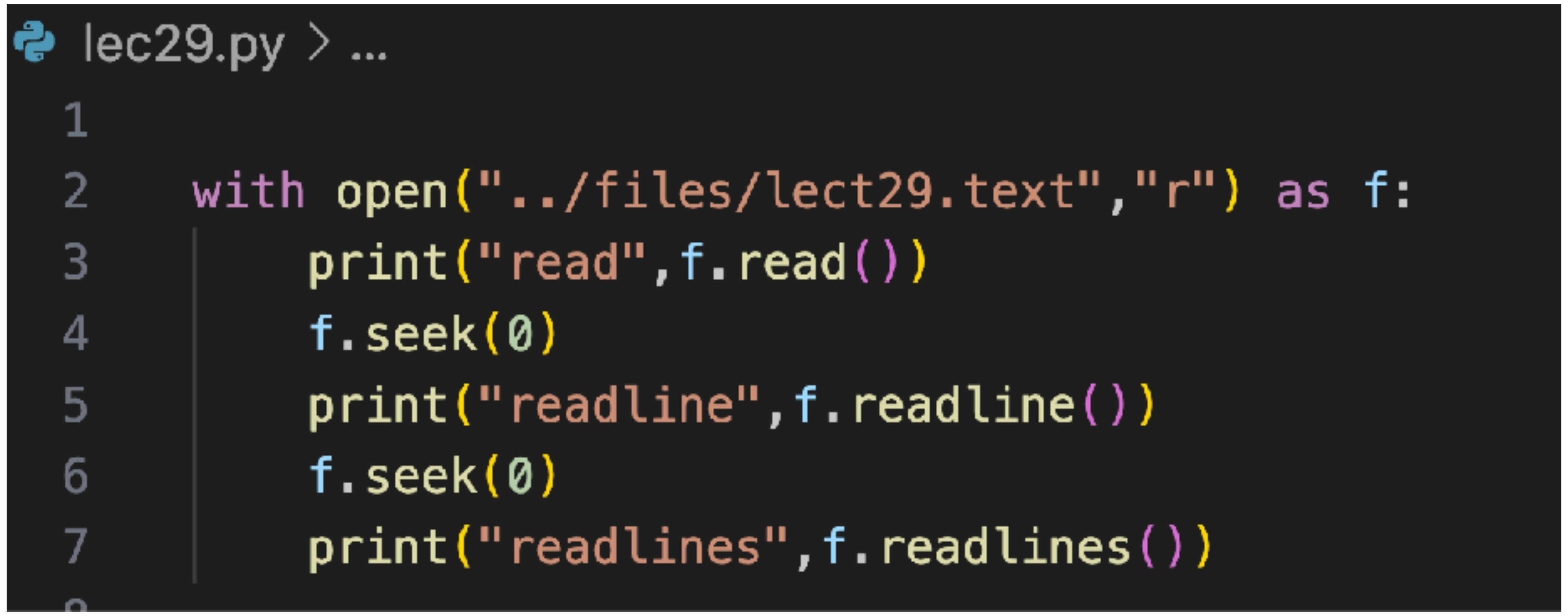
Reading Files



lec29.py > ...

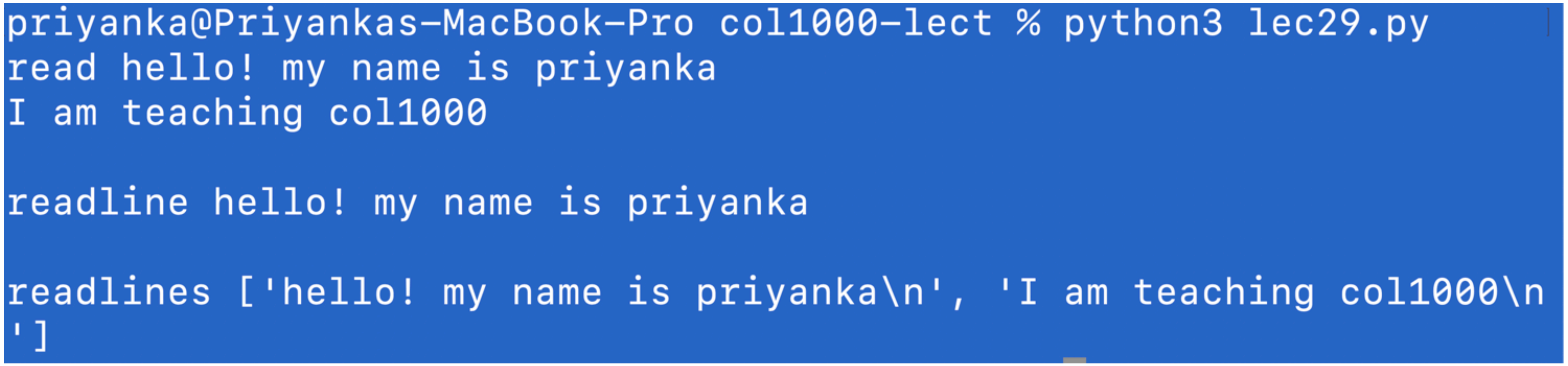
```
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    f.seek(0)
    print("readlines",f.readlines())
```

Reading Files



```
lec29.py > ...

with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    f.seek(0)
    print("readlines",f.readlines())
```

Output



```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
read hello! my name is priyanka
I am teaching col1000

readline hello! my name is priyanka

readlines ['hello! my name is priyanka\n', 'I am teaching col1000\n
']
```

Writing Files

Method	Input Type	Adds Newline Automatically?	Typical Use
write()	single string	❌ No	write single line or message
writeline()	—	❌ (does not exist)	—
writelines()	list/iterable of strings	❌ No	write multiple lines at once

```
lines = ["Hello\n", "World\n", "Python\n"]
with open("sample.txt", "w") as f:
    f.writelines(lines)
```

```
with open("sample.txt", "w") as f:
      f.write(“Hello\n”)
      f.write(“World\n”)
      f.write(“Python\n”)
```

```
Hello
World
Python
```