COL1000
Introduction to Programming

Priyanka Golia

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

Opening and Closing Files

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

Opening and Closing Files

iterable

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

Opening and Closing Files

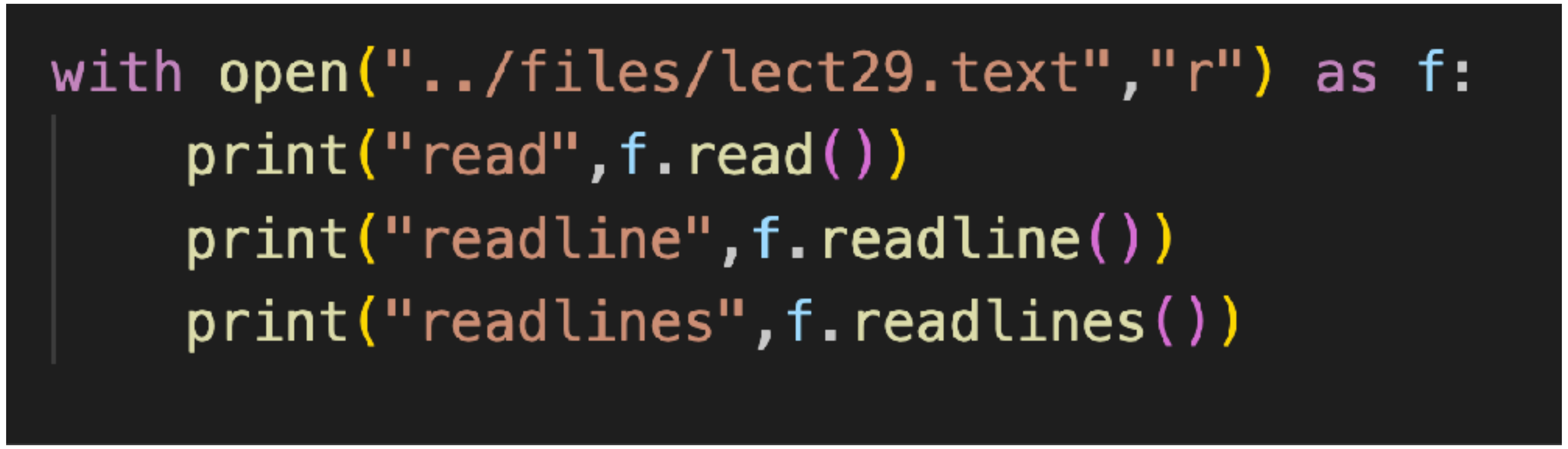
iterable

$\uparrow$

```
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'`.

Reading Files



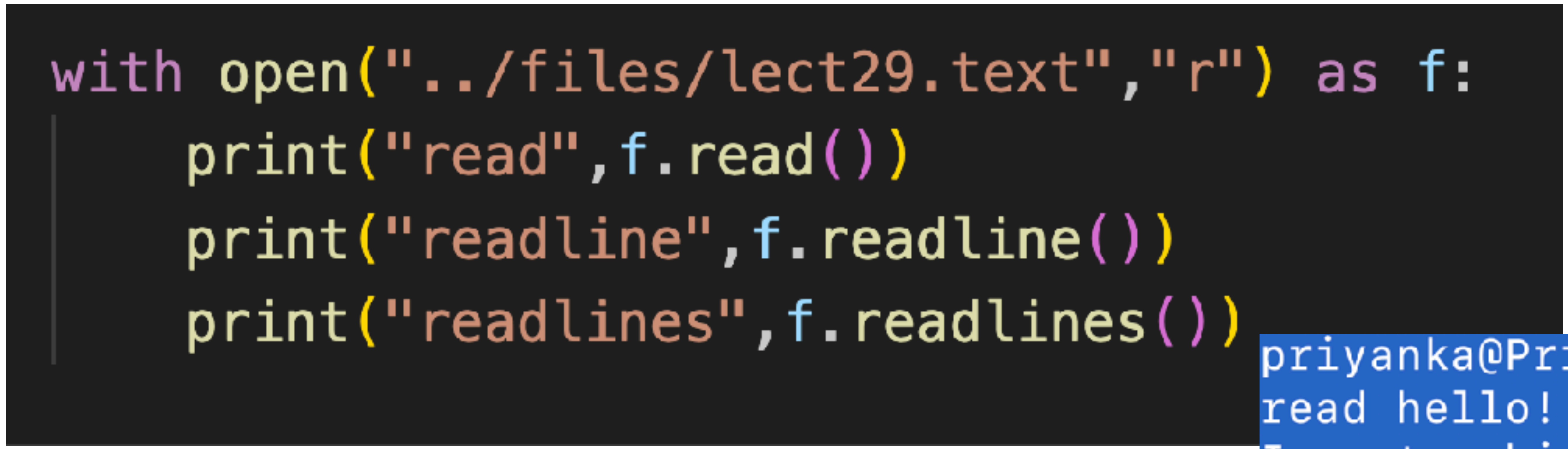
```
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

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.

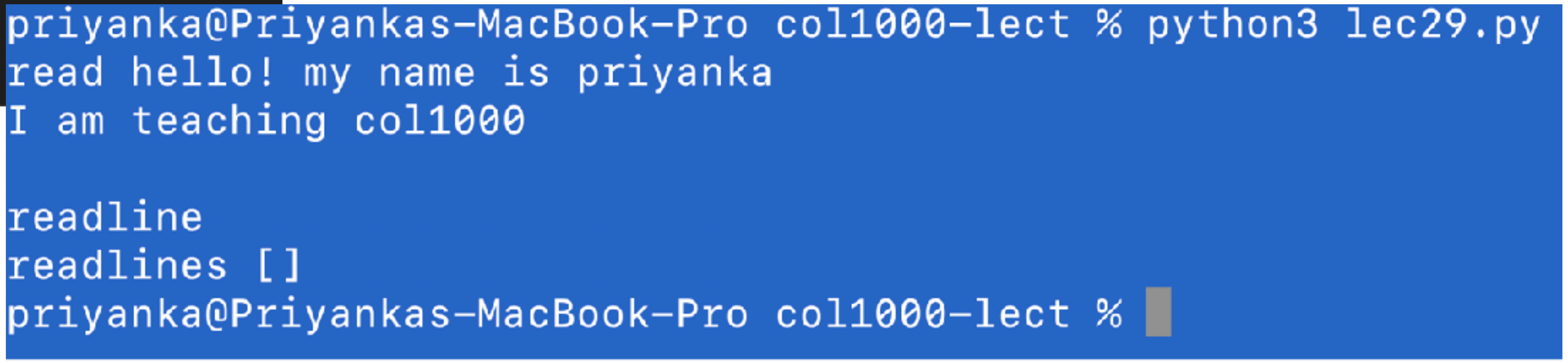
f.readlines() — Reads all lines at once.Returns a list of strings, one per line (each ending with \n).

Reading Files



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

Output



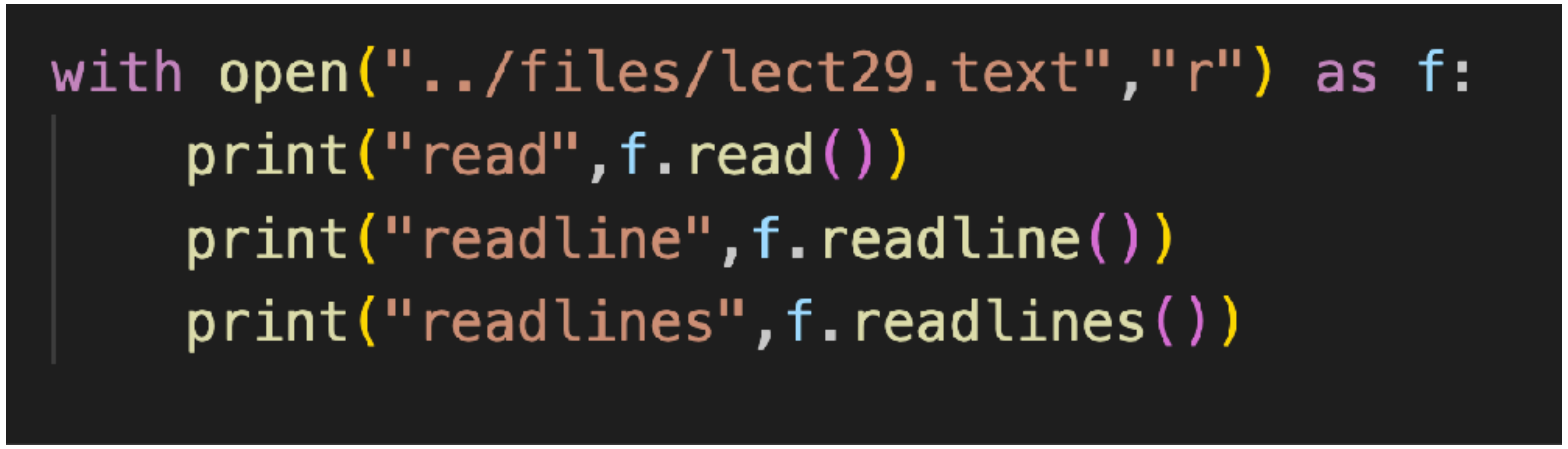
```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
read hello! my name is priyanka
I am teaching col1000

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.

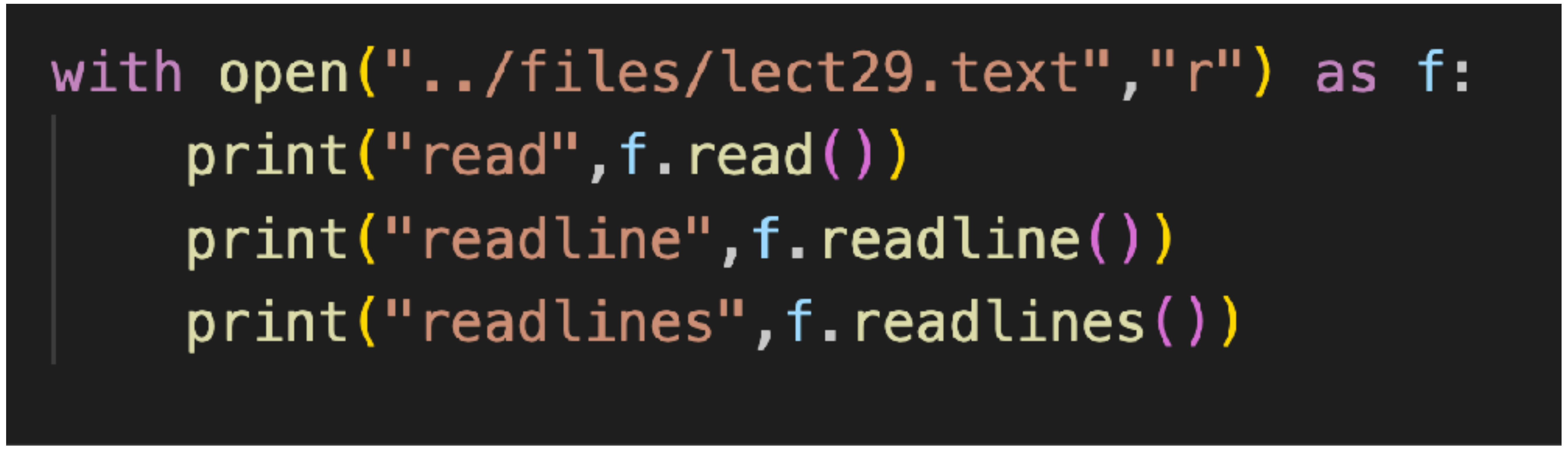
f.readlines() — Reads all lines at once.Returns a list of strings, one per line (each ending with \n).

Reading Files



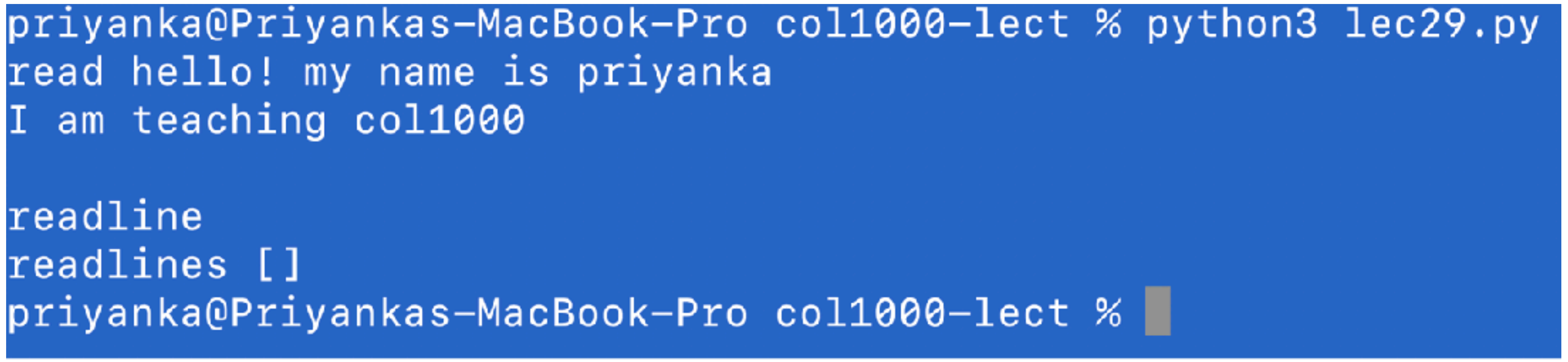
```
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Reading Files



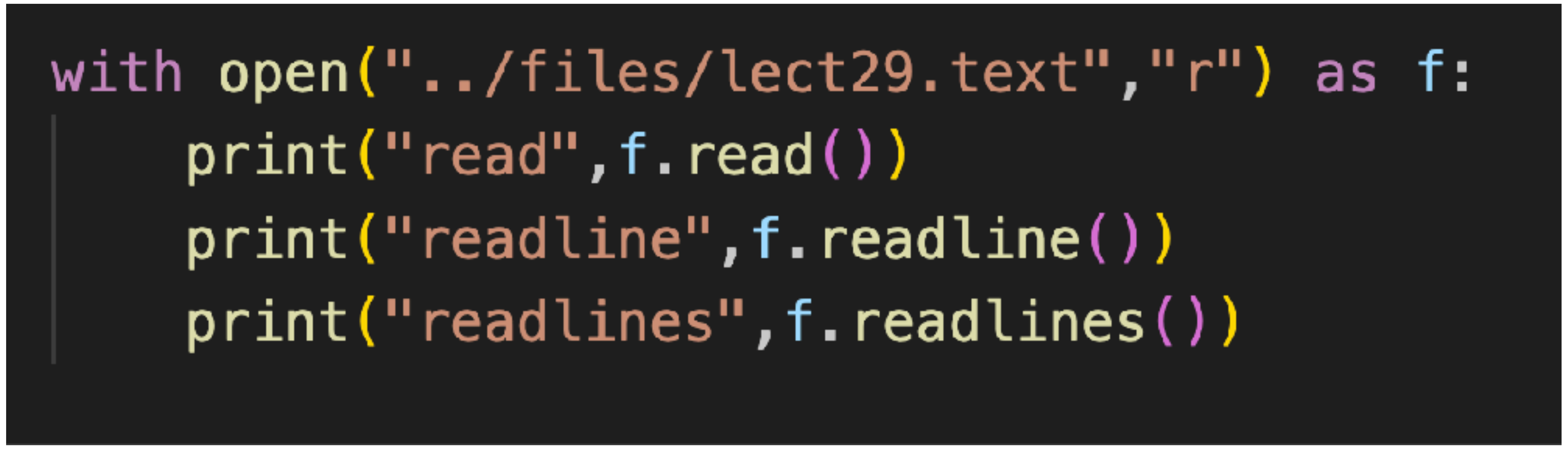
```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Output



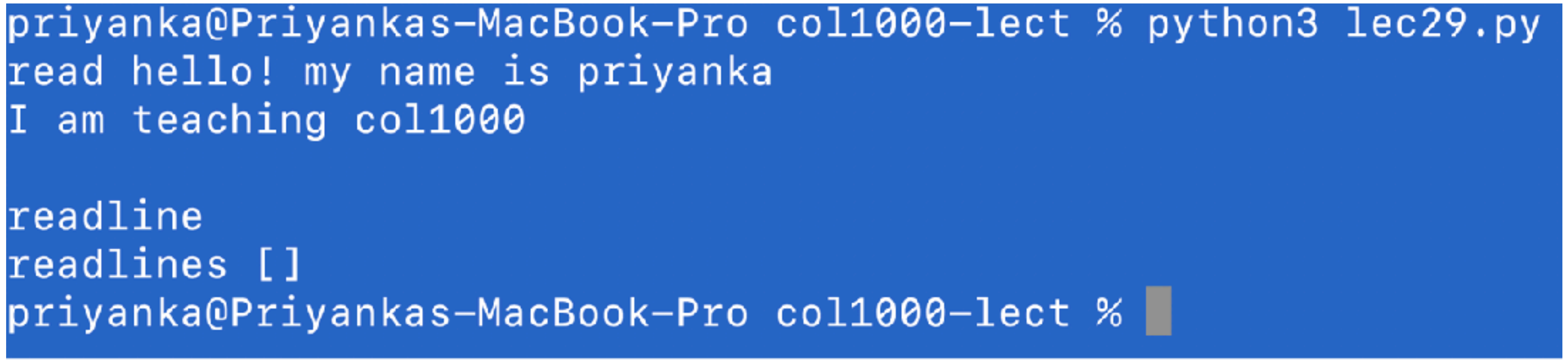
```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
read hello! my name is priyanka
I am teaching col1000

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

Reading Files



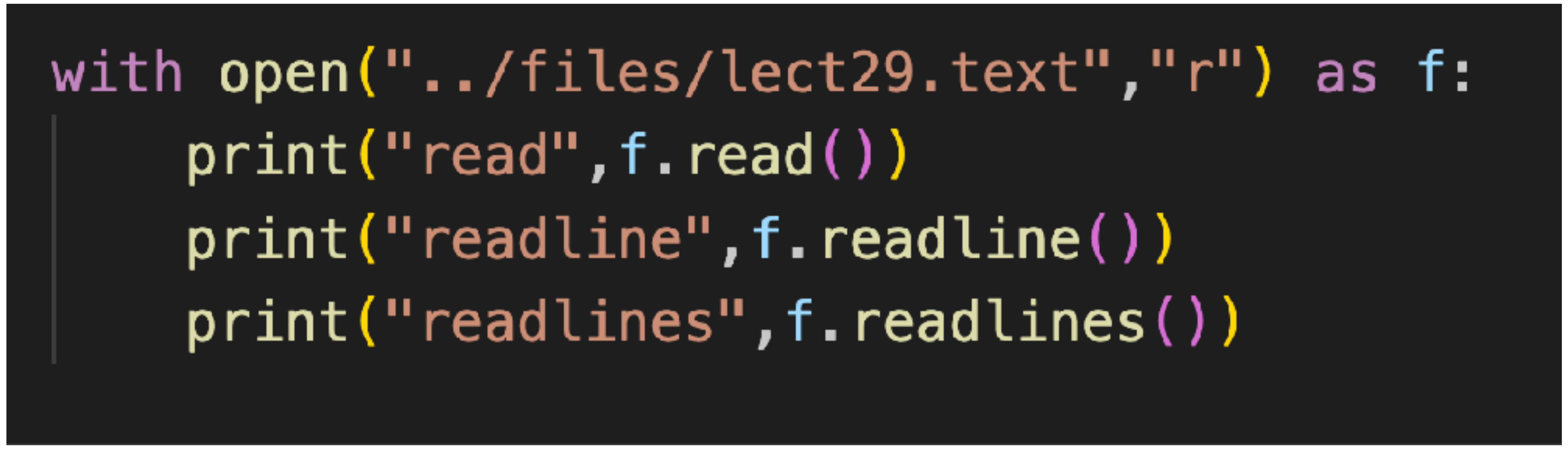
```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

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.

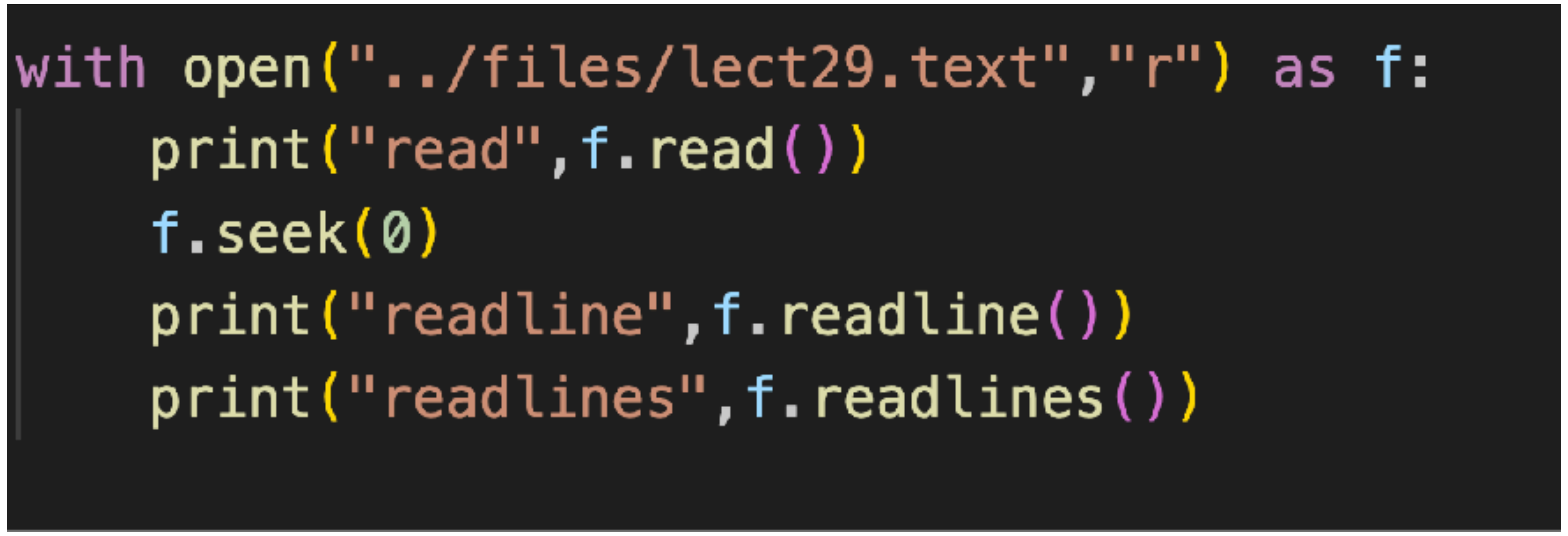


```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    print("readline",f.readline())
    print("readlines",f.readlines())
```

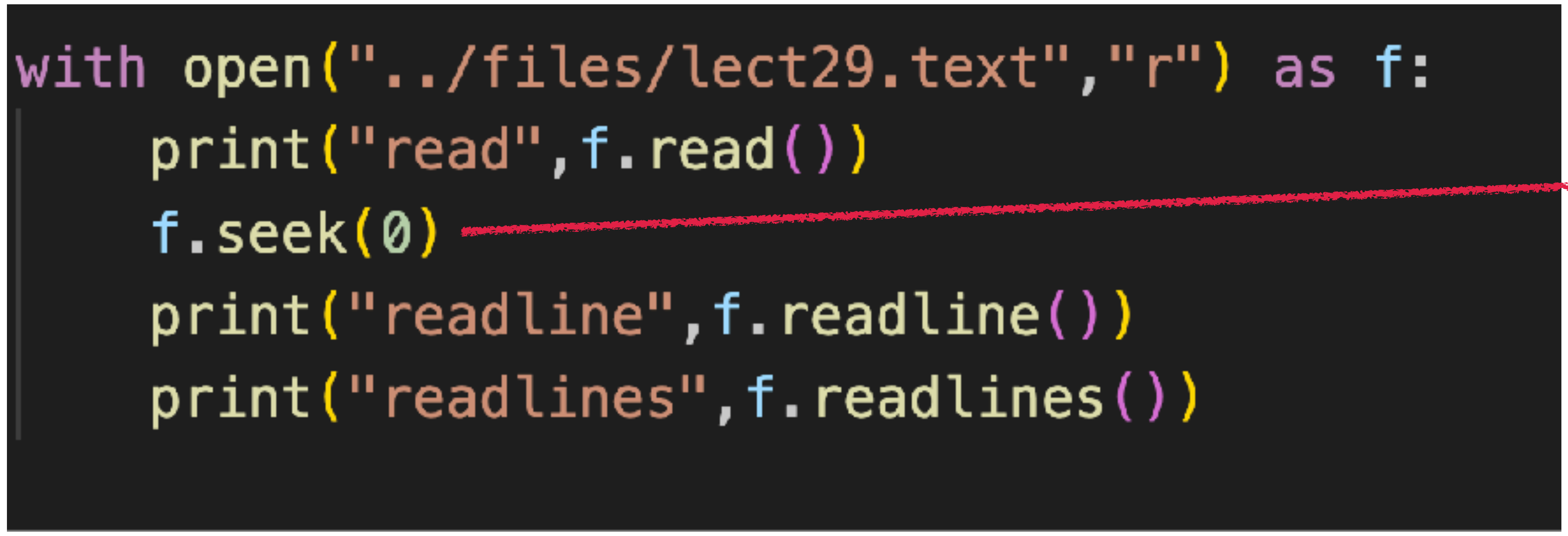
1st print — entire data
2nd print — empty string
3rd print — empty list.

Method	Expected return type	At EOF returns	Reason
f.read()	str	‘'	“nothing left to read” — empty text
f.readline()	str	‘'	“no next line” — empty string
f.readlines()	list[str]	[]	“no more lines” — empty list

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())
```

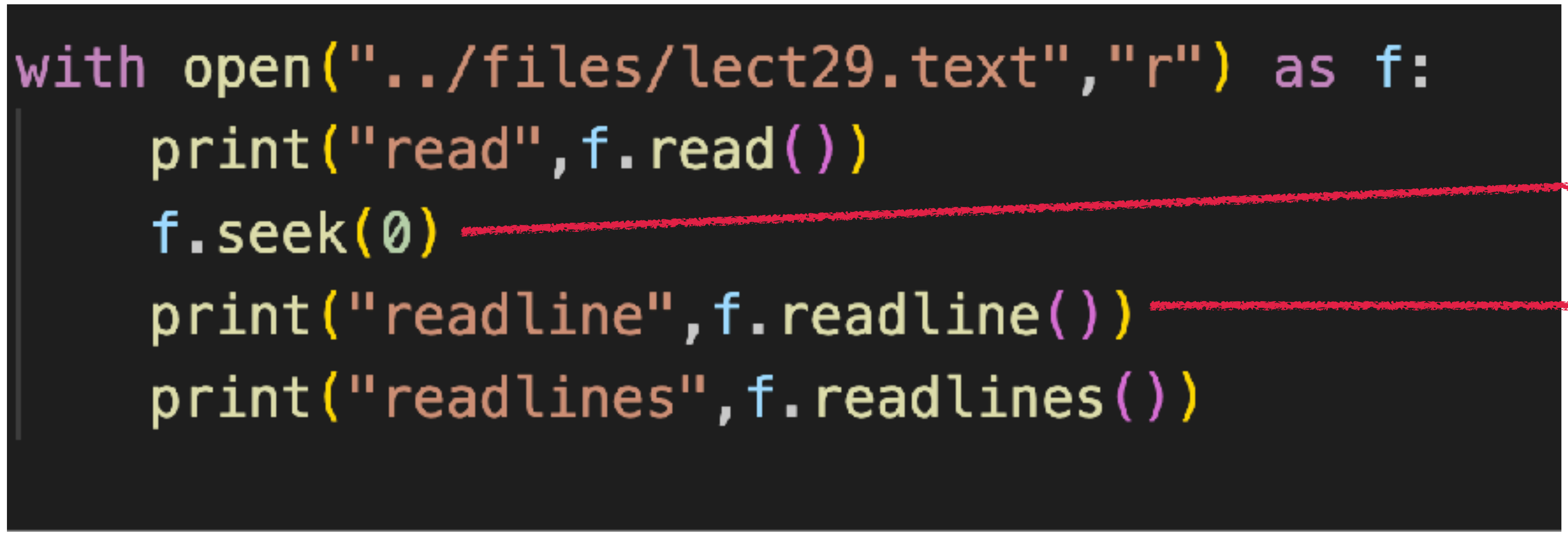
Reading Files



```
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

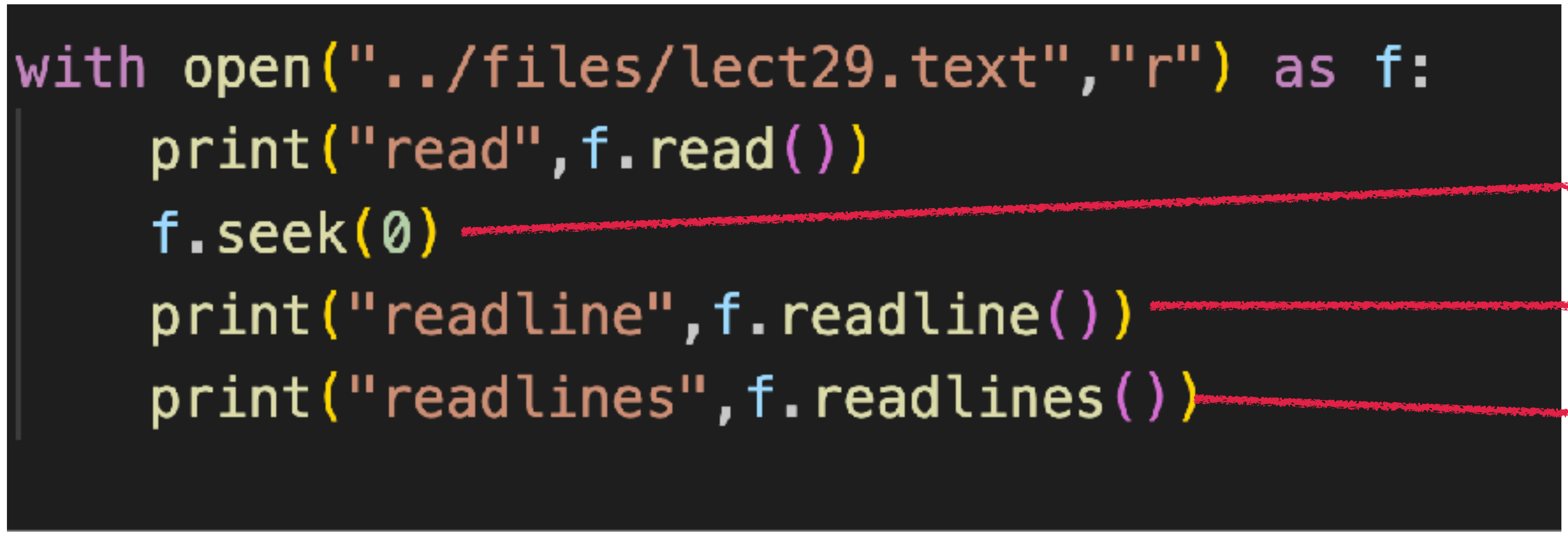


```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Resetting the pointer to position 0.

Returns the current line (a single line)

Reading Files



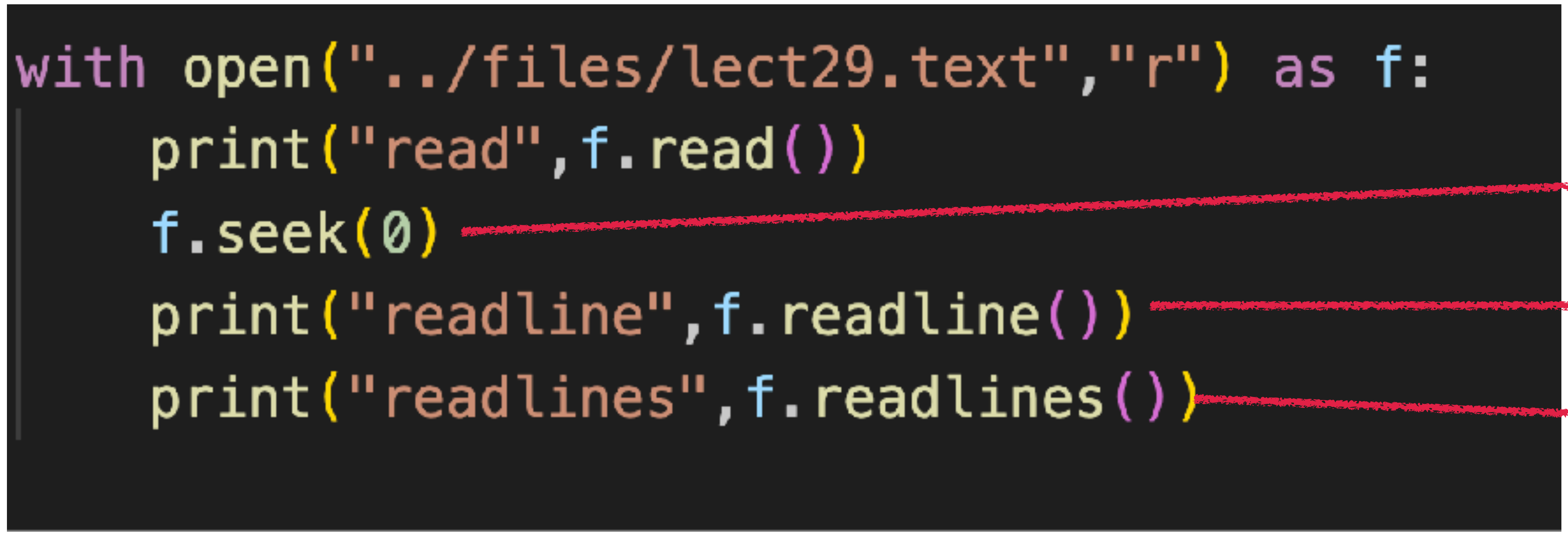
```
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Resetting the pointer to position 0.

Returns the current line (a single line)

Returns the list of lines — from the pointer position.

Reading Files



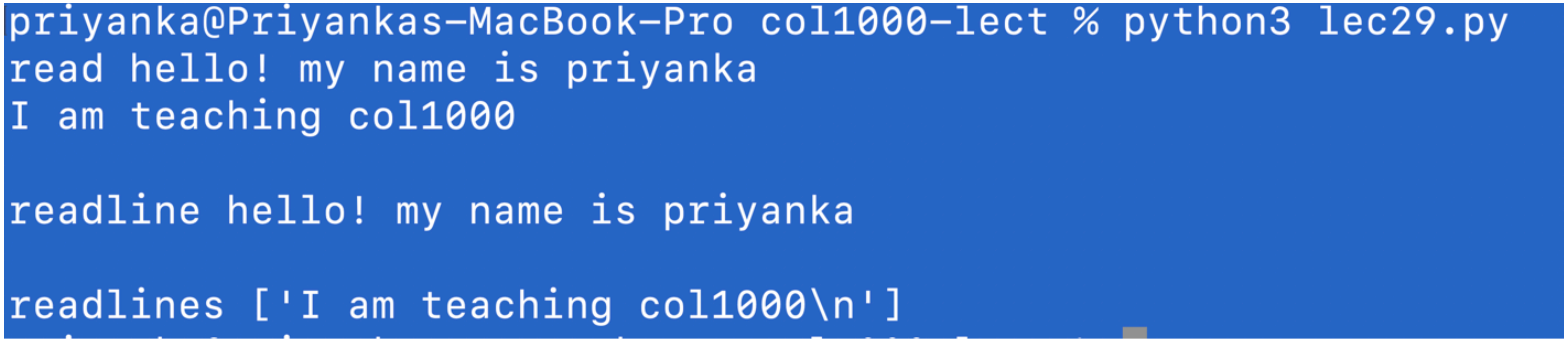
```python
with open("../files/lect29.text","r") as f:
    print("read",f.read())
    f.seek(0)
    print("readline",f.readline())
    print("readlines",f.readlines())
```

Resetting the pointer to position 0.

Returns the current line (a single line)

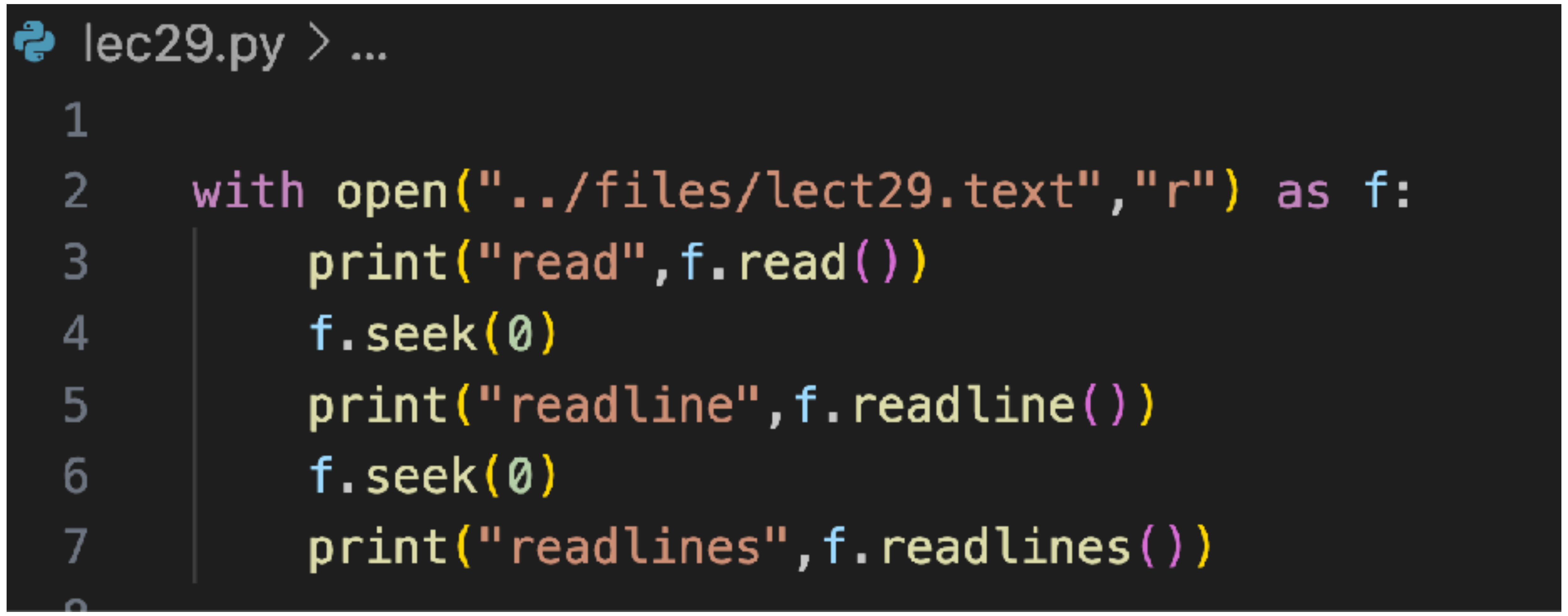
Returns the list of lines — from the pointer position.

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']
```

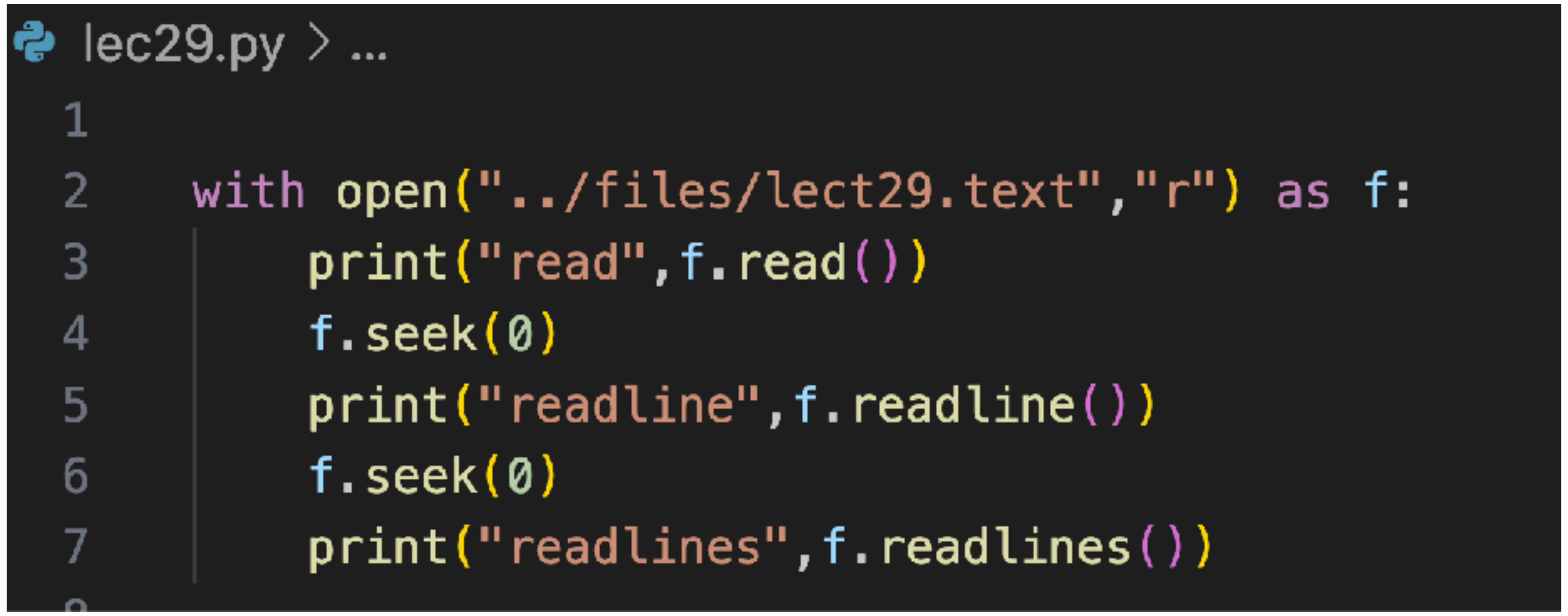
Reading Files



lec29.py > ...

```python

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())
```

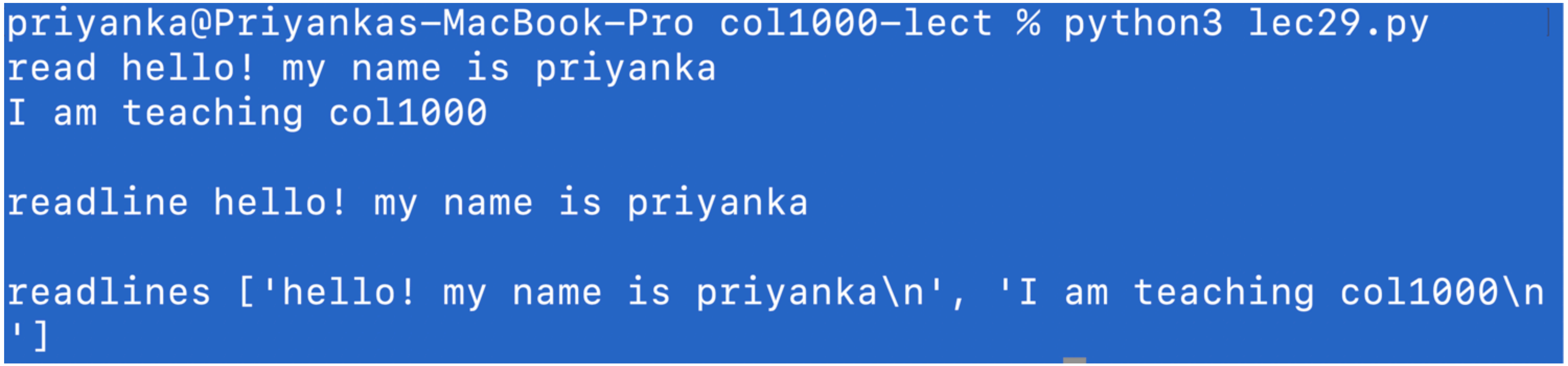
Reading Files



lec29.py > ...

```python

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
```

<file_object>.seek()

When we read or write a file in Python, there is an **internal cursor** (also called the **file pointer**) that keeps track of **where** in the file the next read or write will happen.

By default, the cursor starts at the beginning (0 position).

f.seek(offset, whence)

Offset — how many bytes/characters to move. It is an integer, can be negative.

Whence -> from where to start counting (default is 0)

whence value	Meaning	Example
`0` (default)	from **beginning** of the file	`f.seek(0)` → move to start
1	from current position	`f.seek(10, 1)` → move 10 bytes ahead
2	from end of file	f.seek(0, 2) → move to **end of file**

<file_object>.seek()

When we read or write a file in Python, there is an **internal cursor** (also called the **file pointer**) that keeps track of **where** in the file the next read or write will happen.

By default, the cursor starts at the beginning (0 position).

f.seek(offset, whence)

Offset — how many bytes/characters to move. It is an integer, can be negative.

Whence -> from where to start counting (default is 0)

`f.seek(0)`

From beginning of the file, move 0 characters.

whence value	Meaning	Example
0 (default)	from **beginning** of the file	`f.seek(0)` → move to start
1	from current position	`f.seek(10, 1)` → move 10 bytes ahead
2	from end of file	f.seek(0, 2) → move to **end of file**

<file_object>.seek()

When we read or write a file in Python, there is an **internal cursor** (also called the **file pointer**) that keeps track of **where** in the file the next read or write will happen.

By default, the cursor starts at the beginning (0 position).

f.seek(offset, whence)

Offset — how many bytes/characters to move. It is an integer, can be negative.

Whence -> from where to start counting (default is 0)

`f.seek(0)`

From beginning of the file, move 0 characters.

whence value	Meaning	Example
0 (default)	from **beginning** of the file	`f.seek(0)` → move to start
1	from current position	`f.seek(10, 1)` → move 10 bytes ahead
2	from end of file	f.seek(0, 2) → move to **end of file**

<file_object>.seek()

When we read or write a file in Python, there is an **internal cursor** (also called the **file pointer**) that keeps track of **where** in the file the next read or write will happen.

By default, the cursor starts at the beginning (0 position).

f.seek(offset, whence)

Offset — how many bytes/characters to move. It is an integer, can be negative.

Whence -> from where to start counting (default is 0)

`f.seek(0)`

From beginning of the file, move 0 characters.

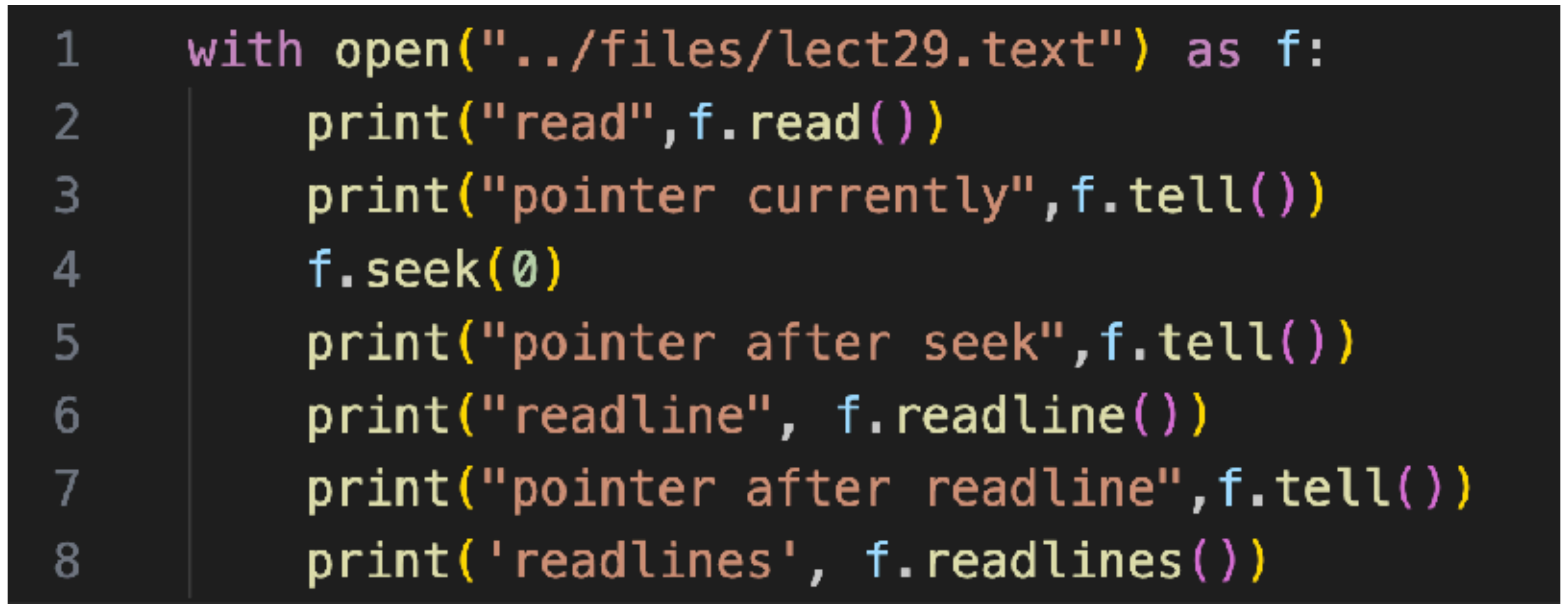
File must be opened in rb mode

whence value	Meaning	Example
0 (default)	from **beginning** of the file	`f.seek(0)` → move to start
1	from current position	`f.seek(10, 1)` → move 10 bytes ahead
2	from end of file	f.seek(0, 2) → move to **end of file**

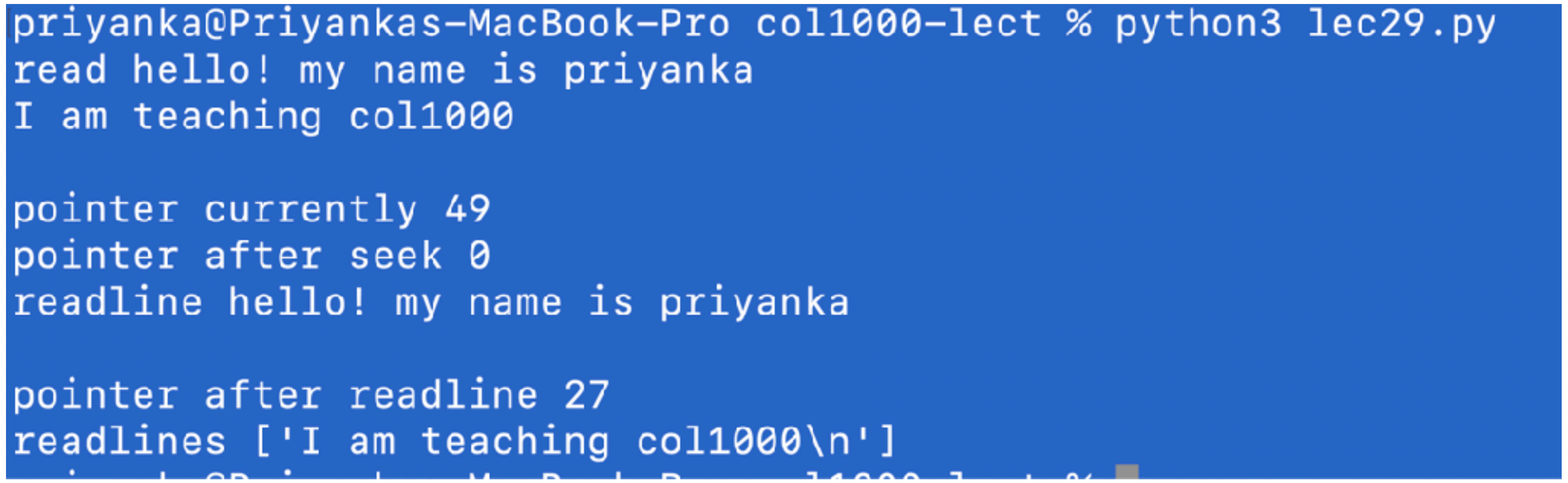
<file_object>.tell()

When we read or write a file in Python, there is an **internal cursor** (also called the **file pointer**) that keeps track of **where** in the file the next read or write will happen.

f.tell() simply tells you that position.



```python
with open("../files/lect29.text") as f:
    print("read",f.read())
    print("pointer currently",f.tell())
    f.seek(0)
    print("pointer after seek",f.tell())
    print("readline", f.readline())
    print("pointer after readline",f.tell())
    print('readlines', f.readlines())
```

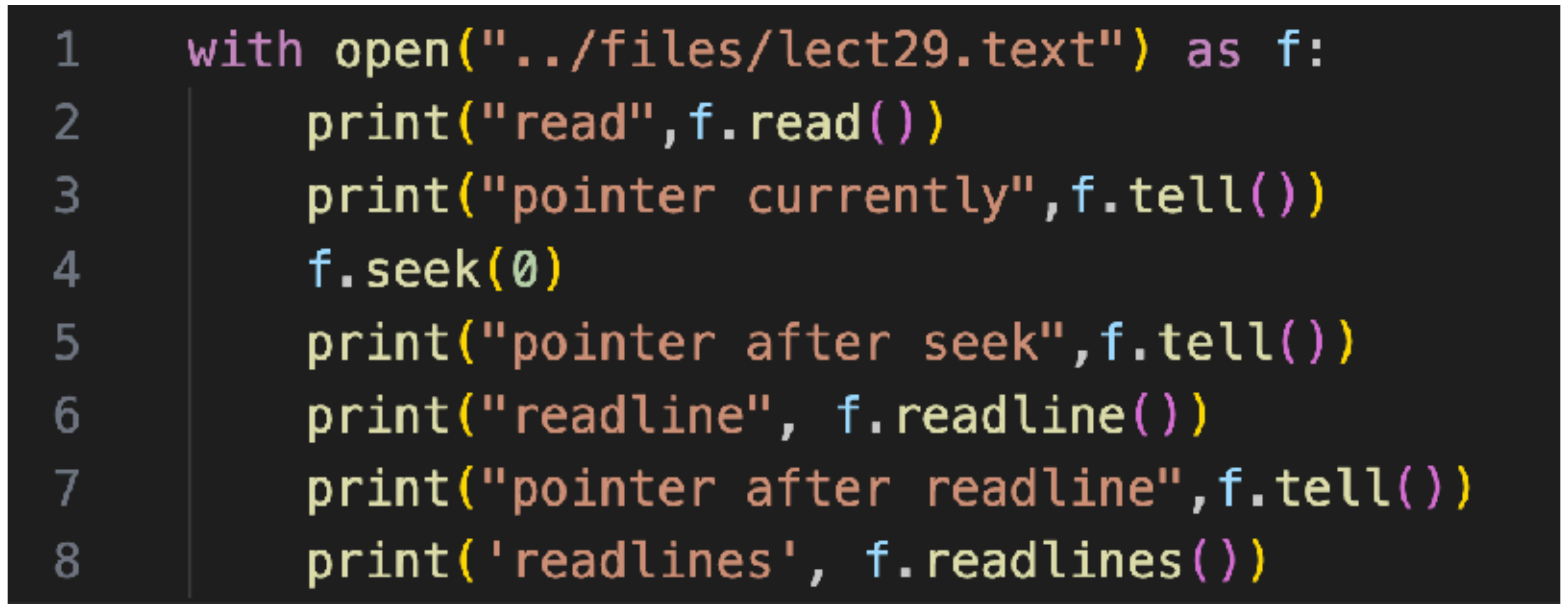


```
priyanka@Priyankas-MacBook-Pro col1000-lect % python3 lec29.py
read hello! my name is priyanka
I am teaching col1000

pointer currently 49
pointer after seek 0
readline hello! my name is priyanka

pointer after readline 27
readlines ['I am teaching col1000\n']
```

<file_object>.tell()

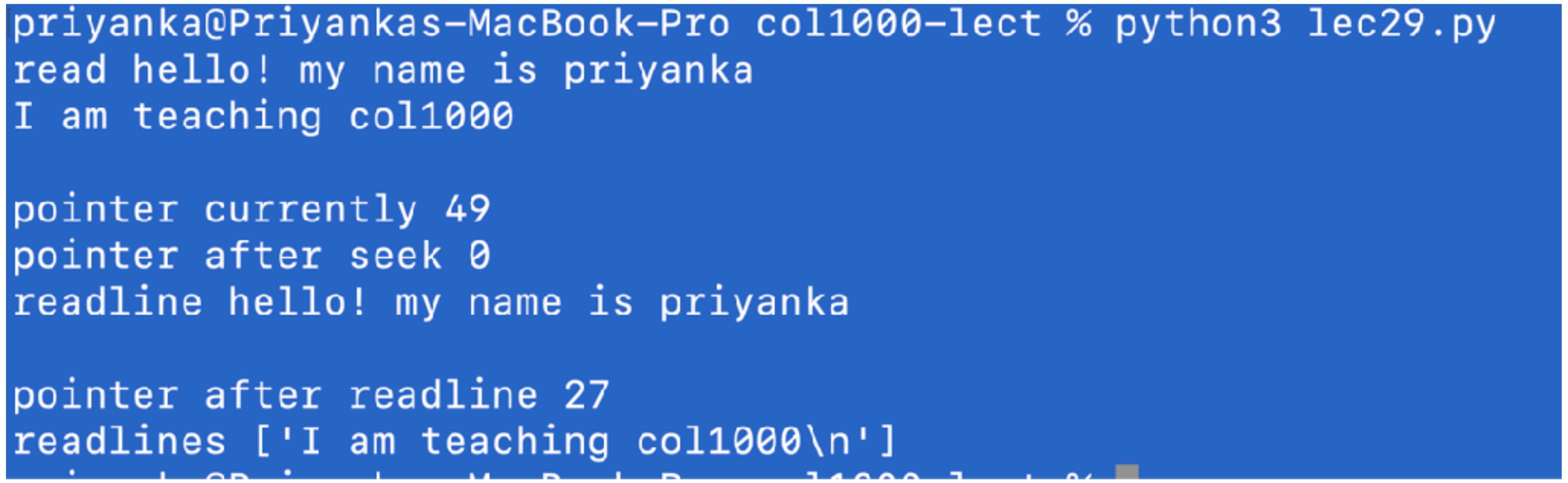
When we read or write a file in Python, there is an **internal cursor** (also called the **file pointer**) that keeps track of **where** in the file the next read or write will happen.

f.tell() simply tells you that position.



```python
with open("../files/lect29.text") as f:
    print("read",f.read())
    print("pointer currently",f.tell())
    f.seek(0)
    print("pointer after seek",f.tell())
    print("readline", f.readline())
    print("pointer after readline",f.tell())
    print('readlines', f.readlines())
```

Output



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

pointer currently 49
pointer after seek 0
readline hello! my name is priyanka

pointer after readline 27
readlines ['I am teaching col1000\n']
```

Other modes

Mode	Read	Write	Create if Missing	Truncate File	Start Position
r+	✅	✅	❌	❌	Start
w+	✅	✅	✅	✅ (clears file)	Start
a+	✅	✅	✅	❌ (append only)	End

File Exceptions and Error Handling

```
try:
    f = open(“../files/lect29.text”, "w")
except FileNotFoundError:
    print("File not found.")
except PermissionError:
    print("You don’t have permission to write to this file.")
except OSError as e:
    print("Other OS-related error:", e)
```

```
BaseException
 └── Exception
      └── OSError
           ├── FileNotFoundError
           ├── PermissionError
           ├── IsADirectoryError
           ├── NotADirectoryError
           └── BlockingIOError
```

File Exceptions and Error Handling

```
try:
    f = open(“../files/lect29.text”, "w")
except FileNotFoundError:
    print("File not found.")
except PermissionError:
    print("You don’t have permission to write to this file.")
except OSError as e:
    print("Other OS-related error:", e)
```

```
To add permissions:
chmod +r <file>
Chmod +r-w <file>
# to remove write permission
```

```
BaseException
 └── Exception
      └── OSError
           ├── FileNotFoundError
           ├── PermissionError
           ├── IsADirectoryError
           ├── NotADirectoryError
           └── BlockingIOError
```

Universal Structured Formats

Universal Structured Formats

```python
import json
# Write out to JSON format file
data = {'name': 'subodh', 'age': 'no'}
with open('person.json', 'w') as f:
    json.dump(data, f)

# Read from  JSON file
with open('person.json', 'r') as f:
    data = json.load(f)
```

JSON (JavaScript Object Notation) is a lightweight format for storing and exchanging data.
It's used everywhere — configuration files, APIs, data transfer, etc.

Universal Structured Formats

```python
import json
# Write out to JSON format file
data = {'name': 'subodh', 'age': 'no'}
with open('person.json', 'w') as f:
    json.dump(data, f)

# Read from  JSON file
with open('person.json', 'r') as f:
    data = json.load(f)
```

JSON (JavaScript Object Notation) is a lightweight format for storing and exchanging data.
It's used everywhere — configuration files, APIs, data transfer, etc.

CSV (Comma-Separated Values). A CSV file is a **plain text file** where each line represents a row, and each value (or column) is separated by a **comma (,)**.

```python
import csv
with open('data.csv', 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        for column in row:
            print(column)
```

Universal Structured Formats

```
import json
# Write out to JSON format file
data = {'name': 'subodh', 'age': 'no'}
with open('person.json', 'w') as f:
    json.dump(data, f)

# Read from  JSON file
with open('person.json', 'r') as f:
    data = json.load(f)
```

JSON (JavaScript Object Notation) is a lightweight format for storing and exchanging data.
It's used everywhere — configuration files, APIs, data transfer, etc.

CSV (Comma-Separated Values). A CSV file is a **plain text file** where each line represents a row, and each value (or column) is separated by a **comma (,)**.

```
import csv
with open('data.csv', 'r') as f:
    reader = csv.reader(f)
    for row in reader:
        for column in row:
            print(column)
```

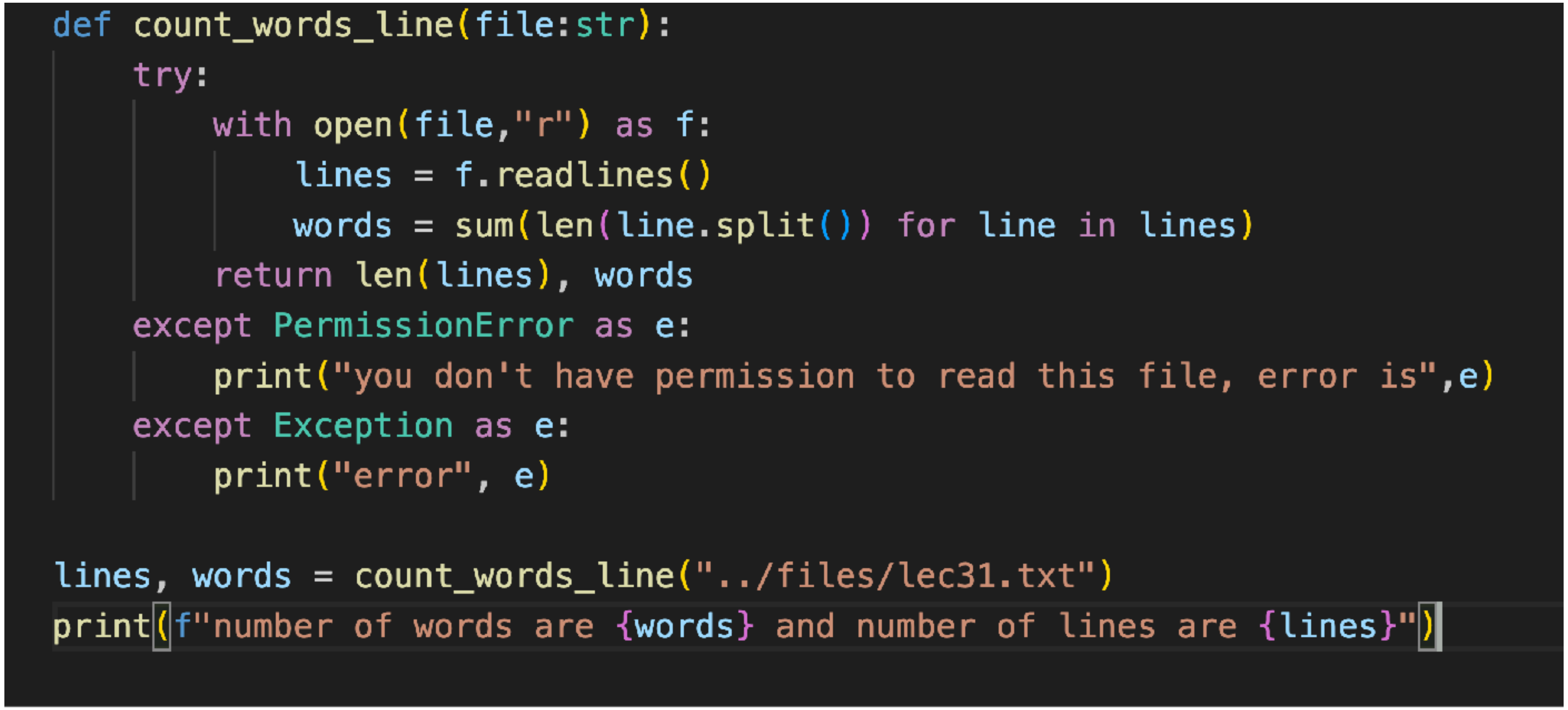
Format	Used For	Structure	Example
JSON	Hierarchical / structured data	nested (dictionary-like)	{"name": "Subodh", "age": 24}
CSV	Tabular data	rows & columns	Subodh,24,Delhi

File Directories

```
import os
current_dir:str = os.getcwd() # Current directory
dir = current_dir+'/data'
if os.path.exists(dir):
    _all = os.listdir(dir)             # Files in folder
    for f in _all:
        if(os.path.isfile(f): print(f'{f} is a file')
```

Write a program to that takes a file name as input, checks for the permission (handles exceptions), and if allowed returns the total number of lines and words in the file content.

Write a program to that takes a file name as input, checks for the permission (handles exceptions), and if allowed returns the total number of lines and words in the file content.



```python
def count_words_line(file:str):
    try:
        with open(file,"r") as f:
            lines = f.readlines()
            words = sum(len(line.split()) for line in lines)
        return len(lines), words
    except PermissionError as e:
        print("you don't have permission to read this file, error is",e)
    except Exception as e:
        print("error", e)

lines, words = count_words_line("../files/lec31.txt")
print(f"number of words are {words} and number of lines are {lines}")
```