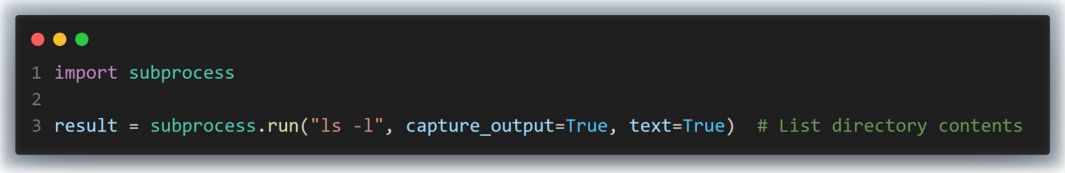
# Overview of the Python `subprocess` Module

❖ The subprocess module in Python allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.
❖ This is particularly useful when you need to interact with system commands or other programs from within your Python script

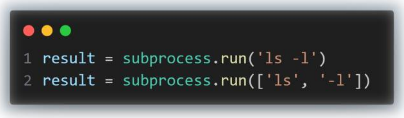
## Running a command:



```python
import subprocess

result = subprocess.run("ls -l", capture_output=True, text=True)  # List directory contents
```

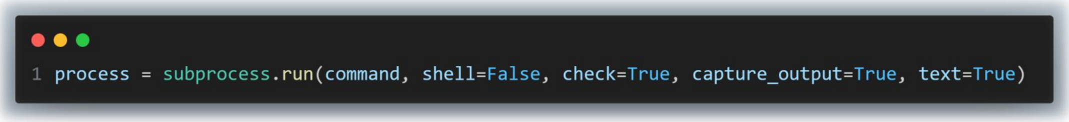
## Command can be :

1- Single string command
2- List of strings command



```python
result = subprocess.run('ls -l')
result = subprocess.run(['ls', '-l'])
```

## Command Parameters:



```python
process = subprocess.run(command, shell=False, check=True, capture_output=True, text=True)
```

### Check:

- specifies whether to raise a CalledProcessError exception if the command returns a non-zero exit status.
- If check is True, and the command fails, it raises an exception.
- If check is False, the function returns normally even if the command fails.

### Capture_output:

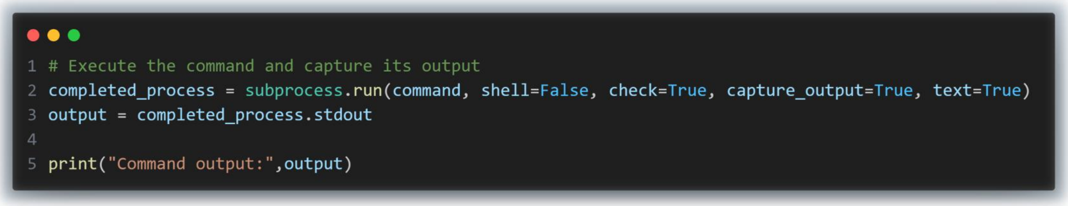
- When set to True, it captures the command's standard output and error streams, returning them as byte strings in the **stdout** and **stderr** attributes of the returned CompletedProcess object.
- This is useful for capturing the output of the command for further processing within your Python script.

### Text:

- determines whether the output from the command should be decoded into text (UTF-8 by default).
- If set to True, the stdout and stderr attributes of the returned CompletedProcess object will contain text strings.
- If set to False, the output will be in binary format.
- Setting it to True is convenient when dealing with text-based output, as it avoids the need for manual decoding.

## Handling output:

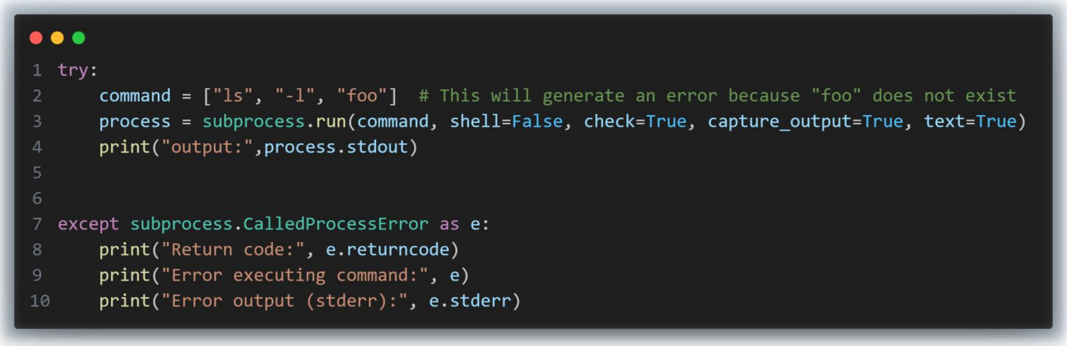
Take the output of command as a string and do whatever you want with it.



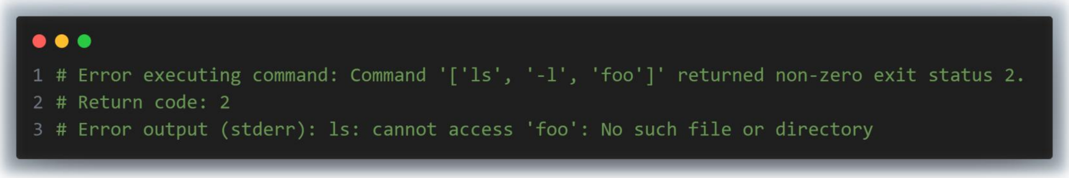
```python
# Execute the command and capture its output
completed_process = subprocess.run(command, shell=False, check=True, capture_output=True, text=True)
output = completed_process.stdout

print("Command output:",output)
```

## Handling Errors with Exception:

Use check=True , to get the error in the command and raise exception



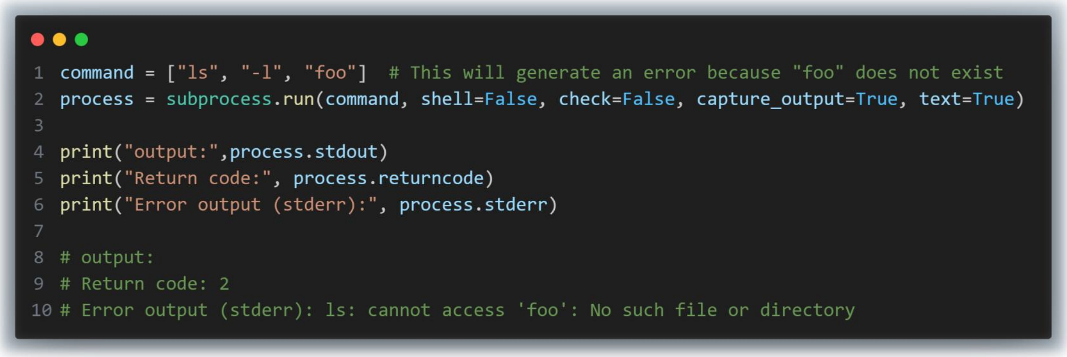
```python
try:
    command = ["ls", "-l", "foo"]  # This will generate an error because "foo" does not exist
    process = subprocess.run(command, shell=False, check=True, capture_output=True, text=True)
    print("output:",process.stdout)


except subprocess.CalledProcessError as e:
    print("Return code:", e.returncode)
    print("Error executing command:", e)
    print("Error output (stderr):", e.stderr)
```



```
# Error executing command: Command '['ls', '-l', 'foo']' returned non-zero exit status 2.
# Return code: 2
# Error output (stderr): ls: cannot access 'foo': No such file or directory
```

## Handling errors without Exceptions :

Use check=False , to get the error in the command without raising exception



```python
command = ["ls", "-l", "foo"]  # This will generate an error because "foo" does not exist
process = subprocess.run(command, shell=False, check=False, capture_output=True, text=True)

print("output:",process.stdout)
print("Return code:", process.returncode)
print("Error output (stderr):", process.stderr)

# output:
# Return code: 2
# Error output (stderr): ls: cannot access 'foo': No such file or directory
```