```
#1.Write a code to read the contents of a file in Python
with open('filename.txt') as f:
    file_contents = f.read()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-1-23b1ac05abf5> in <cell line: 2>()
      1 #1.Write a code to read the contents of a file in Python
----> 2 with open('filename.txt') as f:
      3     file_contents = f.read()

FileNotFoundError: [Errno 2] No such file or directory: 'filename.txt'
```

Next steps: Explain error

```
#2.Write a code to write to a file in Python

with open("my_file.txt", "w") as file:
    file.write("Hello, world!\n")
    file.write("This is a line written to the file.\n")

print("Data written to the file successfully!")
```

```
Data written to the file successfully!
```

```
#3.Write a code to append to a file in Python
with open('my_file.txt', 'a') as file:
    file.write('whatever you want to write here (in append mode) here.')
```

```
#4. Write a code to read a binary file in Python
file = open("example.bin", "rb")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-4-83c3934b5c38> in <cell line: 2>()
      1 #4. Write a code to read a binary file in Python
----> 2 file = open("example.bin", "rb")

FileNotFoundError: [Errno 2] No such file or directory: 'example.bin'
```

Next steps: Explain error

```
#5.What happens if we don't use `with` keyword with `open` in python?
"""If you don’t use the 'with' keyword with 'open' , we’ll need to manually close the file using the .close() """
```

```
'If you don’t use the 'with' keyword with 'open' , we’ll need to manually close the
file using the .close() '
```

```
#6. Explain the concept of buffering in file handling and how it helps in improving read and write operations
"""Buffering provides a temporary storage area, called buffer.
 Buffer can store data before it is sent to or retrieved from the storage device.
 When the buffer is fully occupied, then data is sent to the storage device in a batch,
  this will reduce the number of access operations required and hence improves the performance of the system.
  buffering hels in 1:Synchronization:Synchronization: Buffers synchronize different devices, improving system performance
                    2:Smoothening:Smoothening: Input/output devices operate at different speeds,
                     but buffering ensures smooth data flow, preventing delays or interruptions"""
```

```
'Buffering provides a temporary storage area, called buffer.\n Buffer can store data
before it is sent to or retrieved from the storage device.\n When the buffer is full
y occupied, then data is sent to the storage device in a batch,\n  this will reduce
the number of access operations required and hence improves the performance of the s
ystem.\n  buffering hels in 1:Synchronization:Synchronization: Buffers synchronize d
```

```
#7.Describe the steps involved in implementing buffered file handling in a programming language of your choice
"""The steps involves in implementing buffered file handling in a programming language is:Open a File: Use the open() function to specif
Read Operations:
Use read(), readline(), or readlines() to read data from the file.
Write Operations:
Use write() or writelines() to write data to the file. Buffering:
When you open a file, you can specify the buffering mode:
0 (unbuffered): No buffering takes place.
1 (line buffered): Data is buffered until a newline is reached or the file is closed.
Any other positive value: Use a buffer of approximately that size (in bytes).
A negative value: System default buffering"""
```

```
'The steps involves in implementing buffered file handling in a programming language
is:Open a File: Use the open() function to specify the file name and the mode in whi
ch you want to open the file.\nRead Operations:\nUse read(), readline(), or readline
s() to read data from the file.\nWrite Operations:\nUse write() or writelines() to w
rite data to the file. Buffering:\nWhen you open a file, you can specify the bufferi
```

```
#8.Write a Python function to read a text file using buffered reading and return its content
def read_text_file(file_path):
    buffer_size = 1024
    result = ""
    try:
        with open(file_path, 'r') as file:
            while (chunk := file.read(buffer_size)) != "":
                result += chunk
    except FileNotFoundError: # Handle the exception if the file is not found, this line is now indented to match the try block
        print(f"Error: The file '{file_path}' was not found.")
        return None
    return result

file_path = "your_text_file.txt"
file_contents = read_text_file(file_path)
print(file_contents)
```

```
Error: The file 'your_text_file.txt' was not found.
None
```

```
#9. What are the advantages of using buffered reading over direct file reading in Python?
"""Buffered reading in Python offers several advantages over direct file reading, especially when working with large files or performing

Efficiency: Buffered reading reduces the number of interactions between Python and the file object. Instead of reading from the raw file
This approach is more efficient, especially for slow I/O devices like hard disks or network file systems.
Performance: When reading large files, using a limited buffer size allows you to process data in smaller chunks.
This can significantly enhance performance and resource efficiency, as it avoids loading the entire file into memory at once"""
```

```
'Buffered reading in Python offers several advantages over direct file reading, espe
cially when working with large files or performing frequent I/O operations:\n\nEffic
iency: Buffered reading reduces the number of interactions between Python and the fi
le object. Instead of reading from the raw filestream every time, a chunk of data is
loaded into memory (the buffer) and consumed as needed. \nThis approach is more effi
cient, especially for slow I/O devices like hard disks or network file systems.\nPer
```

```
#10.Write a Python code snippet to append content to a file using buffered writing
def append_text_to_file(file_path, text_to_append):
    try:
        with open(file_path, 'a') as file:
            file.write(text_to_append + '\\n')
            print(f"Text appended to {file_path} successfully.")
    except Exception as e:
        print(f"Error: {e}")
file_path = 'example.txt'
text_to_append = 'This is a new line of text.'
append_text_to_file(file_path, text_to_append)
```

```
Text appended to example.txt successfully.
```

```
#11. Write a Python function that demonstrates the use of close() method on a file
def read_file_with_close(file_path):
    try:
        file = open(file_path, 'r')
        contents = file.read()
        print(contents)
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        if 'file' in locals():
            file.close()
            print("File has been closed.")

file_path = 'path/to/your/file.txt'
read_file_with_close(file_path)
```

```
An error occurred: [Errno 2] No such file or directory: 'path/to/your/file.txt'
```

```
#12. Create a Python function to showcase the detach() method on a file object.
def demonstrate_detach():
    with open("example.txt", "rb") as file:
        buffer = file.detach()
        print(buffer.read(5))
demonstrate_detach()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-da0b659ddf3c> in <cell line: 6>()
      4         buffer = file.detach()
      5         print(buffer.read(5))
----> 6 demonstrate_detach()

<ipython-input-2-da0b659ddf3c> in demonstrate_detach()
      1 #12. Create a Python function to showcase the detach() method on a file
object.
      2 def demonstrate_detach():
----> 3     with open("example.txt", "rb") as file:
      4         buffer = file.detach()
      5         print(buffer.read(5))

FileNotFoundError: [Errno 2] No such file or directory: 'example.txt'
```

Next steps: Explain error

```
#13.Write a Python function to demonstrate the use of the seek() method to change the file position
def demonstrate_seek():
    with open("sample.txt", "r") as file_handle:
        file_handle.seek(10)
        print(file_handle.readline())
demonstrate_seek()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-3-1d8d23d27e9c> in <cell line: 6>()
      4         file_handle.seek(10)
      5         print(file_handle.readline())
----> 6 demonstrate_seek()

<ipython-input-3-1d8d23d27e9c> in demonstrate_seek()
      1 #13.Write a Python function to demonstrate the use of the seek() method to
change the file position
      2 def demonstrate_seek():
----> 3     with open("sample.txt", "r") as file_handle:
      4         file_handle.seek(10)
      5         print(file_handle.readline())

FileNotFoundError: [Errno 2] No such file or directory: 'sample.txt'
```

Next steps: Explain error

```
#14. Create a Python function to return the file descriptor (integer number) of a file using the fileno() method
def get_file_descriptor(file_path):
    try:
        with open(file_path, 'r') as file:
            return file.fileno()
    except OSError:
        print(f"Error: Could not get file descriptor for {file_path}")
        return None
file_path = 'example.txt'
descriptor = get_file_descriptor(file_path)
if descriptor is not None:
    print(f"File descriptor for '{file_path}': {descriptor}")
```

```
Error: Could not get file descriptor for example.txt
```

```
#15.Write a Python function to return the current position of the file's object using the tell() method
with open('your_file.txt', 'r') as file:
    position = file.tell()
    print("Current position:", position)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-5-bd52ace692d2> in <cell line: 2>()
      1 #15.Write a Python function to return the current position of the file's
object using the tell() method
----> 2 with open('your_file.txt', 'r') as file:
      3     position = file.tell()
      4     print("Current position:", position)
      5

FileNotFoundError: [Errno 2] No such file or directory: 'your_file.txt'
```

Next steps: Explain error

```
#16. Create a Python program that logs a message to a file using the logging module
import logging
def setup_logging(log_file):
    logging.basicConfig(
        filename=log_file,
        level=logging.INFO,
        format='%(asctime)s - %(levelname)s - %(message)s',
        datefmt='%Y-%m-%d %H:%M:%S'   )

def log_message(message):
    logging.info(message)
def main():
    log_file = 'path/to/your/logfile.log'
    setup_logging(log_file)
    log_message("This is a sample log message.")
    print(f"Message logged to {log_file}")
if __name__ == "__main__":
    main()
```

```
Message logged to path/to/your/logfile.log
```

```
#17.Explain the importance of logging levels in Python's logging module
"""Logging levels in Python’s logging module allow you to control the verbosity and importance of the logged messages.
the importance is :Granularity and Filtering:
Logging levels act as labels for log entries, helping manage the granularity of information
Severity and Prioritization:
Each logging level has a specific purpose and severity associated with it.
The available levels include
Flexible Control:
You can configure logging levels for different parts of your application.
Fine-tuning the levels allows you to focus on relevant information during development, debugging, or production.
"""
```

```
'Logging levels in Python’s logging module allow you to control the verbosity and im
portance of the logged messages.\nthe importance is :Granularity and Filtering:\nLog
ging levels act as labels for log entries, helping manage the granularity of informa
tion\nSeverity and Prioritization:\nEach logging level has a specific purpose and se
verity associated with it.\nThe available levels include\nFlexible Control:\nYou can
```

```
#18.Create a Python program that uses the debugger to find the value of a variable inside a loop
import pdb
```

```
def calculate_squares(n):
    squares = []
    for i in range(n):
        square = i ** 2
        squares.append(square)
        pdb.set_trace()  # Set a breakpoint here to inspect the value of 'square'
    return squares

if __name__ == "__main__":
    num = 5
    result = calculate_squares(num)
    print(f"Squares of numbers from 0 to {num-1}: {result}")
```

```
PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "/usr/local/lib/python3.10/dist-packages/IPython/core/debugger.py", line 1075, in cmdloop
    sys.settrace(None)

--KeyboardInterrupt--

KeyboardInterrupt: Interrupted by user
> <ipython-input-8-47e9749fa31e>(5)calculate_squares()
      3 def calculate_squares(n):
      4     squares = []
----> 5     for i in range(n):
      6         square = i ** 2
      7         squares.append(square)

--KeyboardInterrupt--

KeyboardInterrupt: Interrupted by user
> <ipython-input-8-47e9749fa31e>(5)calculate_squares()
      3 def calculate_squares(n):
      4     squares = []
----> 5     for i in range(n):
      6         square = i ** 2
      7         squares.append(square)

ipdb> 56
56
--KeyboardInterrupt--

KeyboardInterrupt: Interrupted by user
> <ipython-input-8-47e9749fa31e>(5)calculate_squares()
      3 def calculate_squares(n):
      4     squares = []
----> 5     for i in range(n):
      6         square = i ** 2
      7         squares.append(square)

--KeyboardInterrupt--

KeyboardInterrupt: Interrupted by user
> <ipython-input-8-47e9749fa31e>(5)calculate_squares()
      3 def calculate_squares(n):
      4     squares = []
----> 5     for i in range(n):
      6         square = i ** 2
      7         squares.append(square)

--KeyboardInterrupt--

KeyboardInterrupt: Interrupted by user
Squares of numbers from 0 to 4: [0, 1, 4, 9, 16]
ipdb>                      ipdb>                      ipdb>                      ipdb>                      ipdb>
```

```
#19.Create a Python program that demonstrates setting breakpoints and inspecting variables using the debugger
def divide(a, b):
    breakpoint()
    result = a / b
    return result
print(divide(5, 0))
```

```
PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "/usr/lib/python3.10/bdb.py", line 336, in set_trace
    sys.settrace(self.trace_dispatch)

> <ipython-input-1-f35d9c949d75>(4)divide()
      2 def divide(a, b):
      3     breakpoint()
----> 4     result = a / b
      5     return result
      6 print(divide(5, 0))

ipdb> 4
4
ipdb> 7
7

PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "/usr/lib/python3.10/bdb.py", line 361, in set_quit
    sys.settrace(None)


PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "/usr/local/lib/python3.10/dist-packages/IPython/core/debugger.py", line 1075,
    sys.settrace(None)

--KeyboardInterrupt--

KeyboardInterrupt: Interrupted by user
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-f35d9c949d75> in <cell line: 6>()
      4     result = a / b
      5     return result
----> 6 print(divide(5, 0))

<ipython-input-1-f35d9c949d75> in divide(a, b)
      2 def divide(a, b):
      3     breakpoint()
----> 4     result = a / b
      5     return result
      6 print(divide(5, 0))

ZeroDivisionError: division by zero
```

Next steps: Explain error

```
#20. Create a Python program that uses the debugger to trace a recursive function
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)

def trace(f):
    f.indent = 0

    def g(x):
        print('|  ' * f.indent + '|--', f.__name__, x)
        f.indent += 1
        value = f(x)
        print('|  ' * f.indent + '|--', 'return', repr(value))
        f.indent -= 1
        return value

    return g

fib = trace(fib)
print(fib(4))
```

```
|-- fib 4
|  |-- fib 3
|  |  |-- fib 2
|  |  |  |-- fib 1
|  |  |  |  |-- return 1
|  |  |  |-- fib 0
|  |  |  |  |-- return 0
|  |  |  |-- return 1
|  |  |-- fib 1
|  |  |  |-- return 1
|  |  |-- return 2
|  |-- fib 2
|  |  |-- fib 1
|  |  |  |-- return 1
|  |  |-- fib 0
|  |  |  |-- return 0
|  |  |-- return 1
|  |-- return 3
3
```

```
#21.Write a try-except block to handle a ZeroDivisionError
try:
    numerator = 10
    denominator = 0
    result = numerator / denominator
except ZeroDivisionError:
    print("Cannot divide by zero")
```

```
Cannot divide by zero
```

```
#22. How does the else block work with try-except
"""in try except the else block is useful for scenarios where you want something to happen only if no exceptions were raised.
 It allows you to handle successful execution without catching exceptions that weren’t directly related to the code being protected by t
```

```
'in try except the else block is useful for scenarios where you want something to ha
ppen only if no exceptions were raised.\n It allows you to handle successful executi
on without catching exceptions that weren’t directly related to the code being prote
```

```
#23. Implement a try-except-else block to open and read a file
try:
    with open("file.txt", "r") as f:
        contents = f.read()
        print(contents)
except IOError:
    print("Could not open/read the file.")
```

```
Could not open/read the file.
```

```
#24. What is the purpose of the finally block in exception handling
"""The finally block in exception handling serves several crucial purposes:
Guaranteed Execution: Regardless of whether an error occurs in the try block, the code within the finally block always runs.
Resource Cleanup: It’s commonly used for tasks like closing files, disconnecting database connections, and freeing up resources
State Restoration: Sometimes, it’s used to restore object states or revert changes made during the try block.
Finalization: Certain actions need to be performed regardless of success or failure"""
```

```
'The finally block in exception handling serves several crucial purposes:\nGuarantee
d Execution: Regardless of whether an error occurs in the try block, the code within
the finally block always runs.\nResource Cleanup: It’s commonly used for tasks like
closing files, disconnecting database connections, and freeing up resources\nState R
```

```
#25.Write a try-except-finally block to handle a ValueError
import math

try:
    x = int(input('Please enter a positive number: '))
    print(f'Square Root of {x} is {math.sqrt(x)}')
except ValueError as ve:
    print(f'You entered {x}, which is not a positive number.')
finally:
    print("The 'try-except' block is finished.")
```

```
Please enter a positive number: 5
Square Root of 5 is 2.23606797749979
The 'try-except' block is finished.
```

```
#26.How multiple except blocks work in Python?
try:

    result = 10 / 0
except ZeroDivisionError:
    print("You cannot divide by zero.")
except FirstException:

    pass
except SecondException:

    pass
except (ThirdException, FourthException, FifthException) as e:

    pass
except Exception:

    pass
```

```
You cannot divide by zero.
```

```
#27.What is a custom exception in Python?
"""In Python, a custom exception is a user-defined exception created by the programmer to handle specific error scenarios in a program.
Here’s how you can define and use custom exceptions.
Defining a Custom Exception: To create a custom exception, you need to define a new class that inherits from the built-in Exception clas
```

```
'In Python, a custom exception is a user-defined exception created by the programmer
to handle specific error scenarios in a program. When an error occurs, Python raises
an exception, which can be caught and handled using try and except blocks. \nHere’s
how you can define and use custom exceptions.\nDefining a Custom Exception: To creat
```

```
#28. Create a custom exception class with a message
class CustomException(Exception):
    def __init__(self, msg='My default message', *args, **kwargs):
        super().__init__(msg, *args, **kwargs)
```

```
#29.Write a code to raise a custom exception in Python
class MyCustomException(Exception):
    def __init__(self, message="This is a custom exception."):
        super().__init__(message)
try:
    raise MyCustomException("Something went wrong!")
except MyCustomException as e:
    print(f"Caught custom exception: {e}")
```

```
Caught custom exception: Something went wrong!
```

```
#30.Write a function that raises a custom exception when a value is negative
def validate_positive(value):
    if value < 0:
        raise ValueError("Value must be positive")
try:
    user_input = float(input("Enter a value: "))
    validate_positive(user_input)
    print(f"The value {user_input} is positive.")
except ValueError as e:
    print(f"Error: {e}")
```

```
#31. What is the role of try, except, else, and finally in handling exceptions
"""In Python, you can handle exceptions using the try/except/else/finally statements. These statements provide a way to catch and handle
```

```
'In Python, you can handle exceptions using the try/except/else/finally statements.
These statements provide a way to catch and handle exceptions, execute specific code
when no exceptions occur, and perform cleanup operations regardless of whether an ex
```

```
#32. How can custom exceptions improve code readability and maintainability
"""Custom exceptions can improve code readability and maintainability by providing a clear and descriptive way to handle specific error
By using custom exceptions, developers can create more resilient and error-tolerant applications, tailoring the exceptions to suit the s
 Additionally, adding explanatory comments to custom exception handling can further enhance code readability during refactoring2."""
```

```
'Custom exceptions can improve code readability and maintainability by providing a c
lear and descriptive way to handle specific error conditions1. \nBy using custom exc
eptions, developers can create more resilient and error-tolerant applications, tailo
ring the exceptions to suit the specific needs of their projects1.\n Additionally, a
```

```
#33.What is multithreading?
"""The concept of multi-threading needs proper understanding of these two terms – a process and a thread.
  A process is a program being executed. A process can be further divided into independent units known as threads.
 A thread is like a small light-weight process within a process. Or we can say a collection of threads is what is known as a process."""
```

```
'The concept of multi-threading needs proper understanding of these two terms – a pr
ocess and a thread.\n  A process is a program being executed. A process can be furth
er divided into independent units known as threads. \n A thread is like a small ligh
t-weight process within a process. Or we can say a collection of threads is what is
```

```
#34. Create a thread in Python?
import threading

class MyThread(threading.Thread):
    def __init__(self, thread_name, thread_ID):
        threading.Thread.__init__(self)
        self.thread_name = thread_name
        self.thread_ID = thread_ID

    def run(self):
        print(f"{self.thread_name} {self.thread_ID}")

thread1 = MyThread("ab", 1000)
thread2 = MyThread("abhijit", 2000)
thread1.start()
thread2.start()
print("Exit")
```

```
ab 1000
abhijit 2000
Exit
```

```
#35.What is the Global Interpreter Lock (GIL) in Python?
"""Python Global Interpreter Lock (GIL) is a type of process lock which is used by python whenever it deals with processes.
 Generally, Python only uses only one thread to execute the set of written statements
 This means that in python only one thread will be executed at a time.
  The performance of the single-threaded process and the multi-threaded process will be the same in python and this is because of GIL in
   We can not achieve multithreading in python because we have global interpreter lock which restricts the threads and works as a single
```

```
'Python Global Interpreter Lock (GIL) is a type of process lock which is used by pyt
hon whenever it deals with processes.\n Generally, Python only uses only one thread
to execute the set of written statements\n This means that in python only one thread
will be executed at a time.\n  The performance of the single-threaded process and th
e multi-threaded process will be the same in python and this is because of GIL in py
```

```
#36. Implement a simple multithreading example in Python
import threading

def calculate_square(number):
    print(f"Square of {number}: {number ** 2}")

def calculate_cube(number):
    print(f"Cube of {number}: {number ** 3}")


t1 = threading.Thread(target=calculate_square, args=(5,))
t2 = threading.Thread(target=calculate_cube, args=(5,))
t1.start()
t2.start()
t1.join()
t2.join()

print("Multithreading example completed.")
```

```
Square of 5: 25
Cube of 5: 125
Multithreading example completed.
```

```
#37.What is the purpose of the `join()` method in threading?
"""The join() method in Python’s threading module serves a crucial purpose: it ensures that the main thread waits for a target thread to
the importance is :Synchronization: When you call join() on a thread, the current thread (usually the main thread) blocks until the spec
Main Thread Coordination: If you have multiple threads running concurrently, calling join() allows the main thread to wait for all other
Daemon Threads: When a thread is in daemon mode (set as a daemon thread), it automatically terminates when the main program exits. Howev
```

```
'The join() method in Python’s threading module serves a crucial purpose: it ensures
that the main thread waits for a target thread to finish before proceeding. \nthe im
portance is :Synchronization: When you call join() on a thread, the current thread
(usually the main thread) blocks until the specified thread completes its execution.
This synchronization ensures that threads finish their work in a coordinated manne
r.\nMain Thread Coordination: If you have multiple threads running concurrently, cal
ling join() allows the main thread to wait for all other threads to finish. Without
```

```
#38.Describe a scenario where multithreading would be beneficial in Python
"""Multithreading in Python can be beneficial in various scenarios.
Improved Performance:
Multithreading allows running multiple tasks simultaneously, which can significantly improve the performance of applications.
Better Responsiveness:
Backend applications benefit from multithreading by running threads after a task is blocked.
Scalability:
When dealing with a large and growing number of users on a server, multithreading improves scalability."""
```

```
'Multithreading in Python can be beneficial in various scenarios. \nImproved Perform
ance:\nMultithreading allows running multiple tasks simultaneously, which can signif
icantly improve the performance of applications.\nBetter Responsiveness:\nBackend ap
plications benefit from multithreading by running threads after a task is blocked.\n
```

```
#39.what is multiprocessing in python
"""Multiprocessing refers to the ability of a system to support more than one processor at the same time.
 Applications in a multiprocessing system are broken to smaller routines that run independently.
  The operating system allocates these threads to the processors improving performance of the system."""
```

```
'Multiprocessing refers to the ability of a system to support more than one processo
r at the same time.\n Applications in a multiprocessing system are broken to smaller
routines that run independently.\n  The operating system allocates these threads to
```

```
#40. How is multiprocessing different from multithreading in Python?
"""In Python, multiprocessing and multithreading are both techniques for concurrent execution, but they have distinct
Multiprocessing is a technique where parallelism in its truest form is achieved. Multiple processes are run across multiple CPU cores, w
 In Python, each process has its own instance of Python interpreter doing the job of executing the instructions."""
```

[illegible]

```
#41. Create a process using the multiprocessing module in Python
import multiprocessing


def worker(num):

    print(f'Worker: {num}')

if __name__ == '__main__':

    processes = []
    for i in range(5):
        p = multiprocessing.Process(target=worker, args=(i,))
        processes.append(p)
        p.start()
    for p in processes:
        p.join()
```