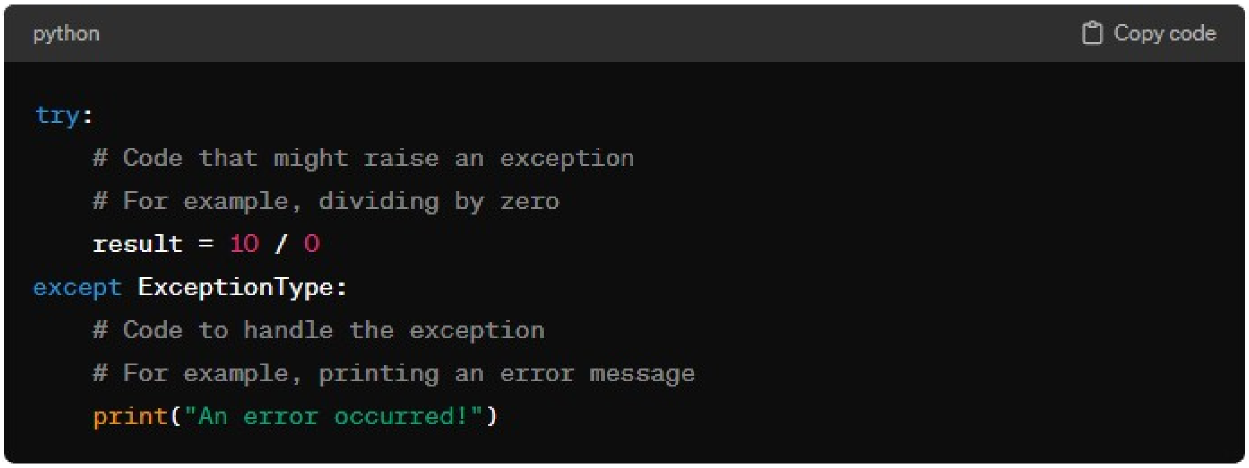
In Python, `try` and `except` blocks are used for exception handling. Exception handling allows you to gracefully manage errors or exceptions that may occur during the execution of your code.

Here's a basic syntax of `try` and `except` blocks:

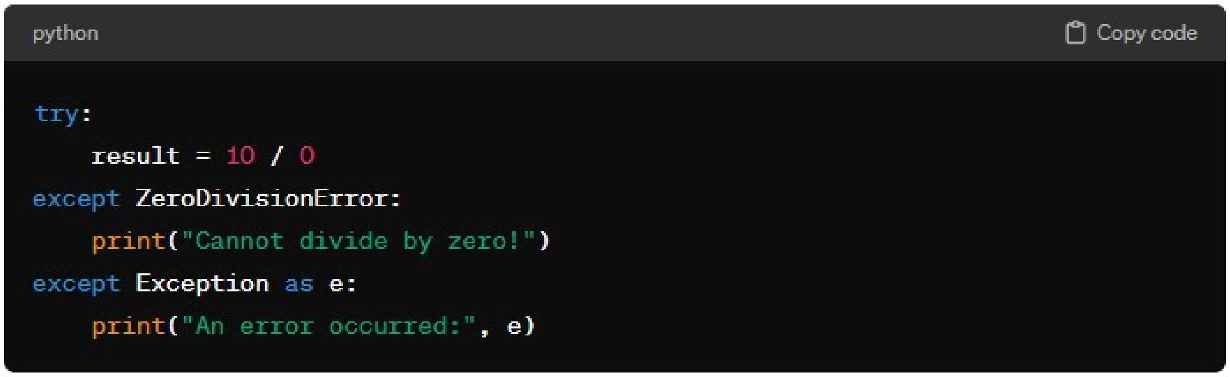


```python
try:
    # Code that might raise an exception
    # For example, dividing by zero
    result = 10 / 0
except ExceptionType:
    # Code to handle the exception
    # For example, printing an error message
    print("An error occurred!")
```

In this code:

- **The `try` block contains the code that might raise an exception.**
- **If an exception of type `ExceptionType` (or any of its subclasses) occurs within the `try` block, Python jumps to the `except` block to handle it.**
- **If no exception occurs, the `except` block is skipped.**
- **You can catch specific types of exceptions by specifying the type after the `except` keyword. If you want to catch any type of exception, you can simply use `except Exception:`.**

Here's an example with more detail:



```python
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
except Exception as e:
    print("An error occurred:", e)
```

In this example:

If a `ZeroDivisionError` occurs (which happens when dividing by zero), the first `except` block is executed, printing "Cannot divide by zero!".

- **If any other type of exception occurs, it's caught by the second `except` block, and it prints the error message along with the exception itself.**

Exception handling allows your program to handle errors gracefully, preventing it from crashing and providing the opportunity to handle errors in a controlled manner.

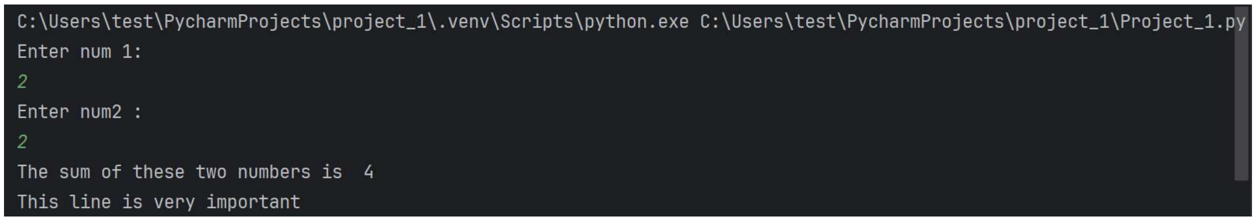
Code:

```
print("Enter num 1: ")
num1 = input()
print("Enter num2 : ")
num2 = input()

try:
    print("The sum of these two numbers is ",
int(num1)+int(num2))
except Exception as e:
    print(e)

print("This line is very important")
```

Output:



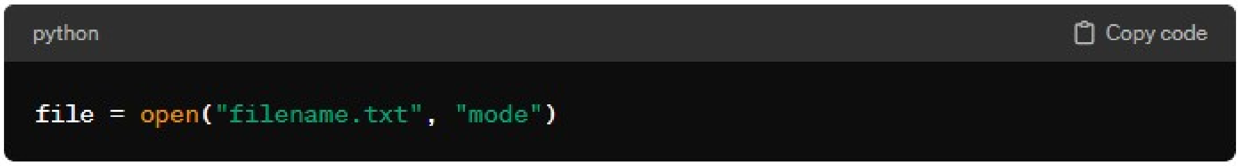
```
C:\Users\test\PycharmProjects\project_1\.venv\Scripts\python.exe C:\Users\test\PycharmProjects\project_1\Project_1.py
Enter num 1:
2
Enter num2 :
2
The sum of these two numbers is  4
This line is very important
```

File I/O (Input/Output) in Python

File I/O (Input/Output) in Python refers to the process of reading from and writing to files on the filesystem. Python provides built-in functions and methods for performing file I/O operations.

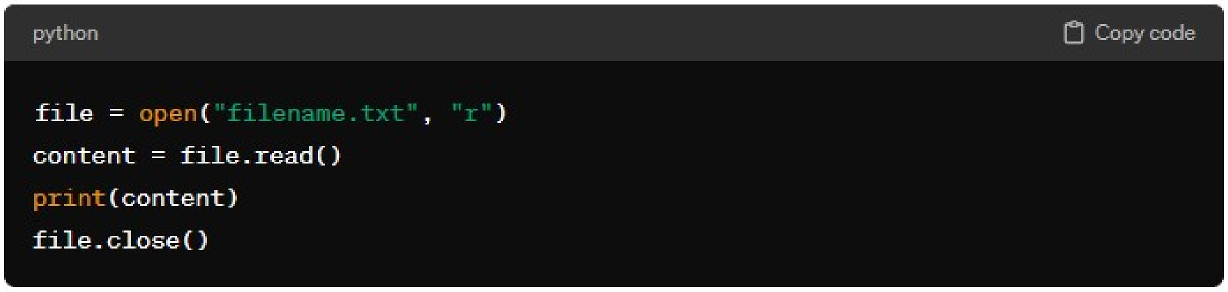
Here are the basic steps for working with files in Python:

1. **Opening a File: To open a file, you use the `open()` function, specifying the file path and the mode in which you want to open the file (e.g., read mode, write mode, append mode, etc.). The basic syntax is:**



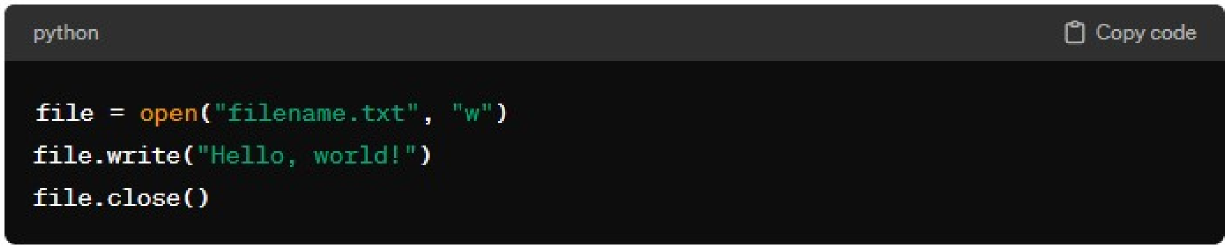
```python
file = open("filename.txt", "mode")
```

Reading from a File**: After opening a file, you can read its contents using various methods like `read()`, `readline()`, or `readlines()`. For example:**



```python
file = open("filename.txt", "r")
content = file.read()
print(content)
file.close()
```

Writing to a File**: To write data to a file, you use the `write()` method. If the file doesn't exist, Python will create it. If it exists, it will overwrite the existing content. For example:**



```python
file = open("filename.txt", "w")
file.write("Hello, world!")
file.close()
```

Appending to a File**: If you want to add content to the end of an existing file without overwriting the existing content, you can open the file in append mode ("a") and use the `write()` method. For example:**

```python
file = open("filename.txt", "a")
file.write("\nThis is a new line.")
file.close()
```

Closing a File**: It's important to close a file after you're done with it to free up system resources and ensure that all data is written to the file. You can use the `close()` method for this purpose, or alternatively, use a context manager (`with` statement) to automatically close the file when you're done with it:**

```python
with open("filename.txt", "r") as file:
    content = file.read()
    print(content)
# File is automatically closed when the block exits
```

Handling Errors**: It's important to handle exceptions that might occur during file I/O operations, especially when dealing with real-world scenarios where files may not exist or permissions may be insufficient. You can use `try` and `except` blocks for this purpose.**

```python
try:
    file = open("filename.txt", "r")
    content = file.read()
    print(content)
    file.close()
except FileNotFoundError:
    print("File not found!")
```

These are the basics of file I/O in Python. It's essential to handle files carefully, especially when dealing with opening, reading, writing, and closing them, to avoid potential issues like resource leaks or data corruption.

Code:

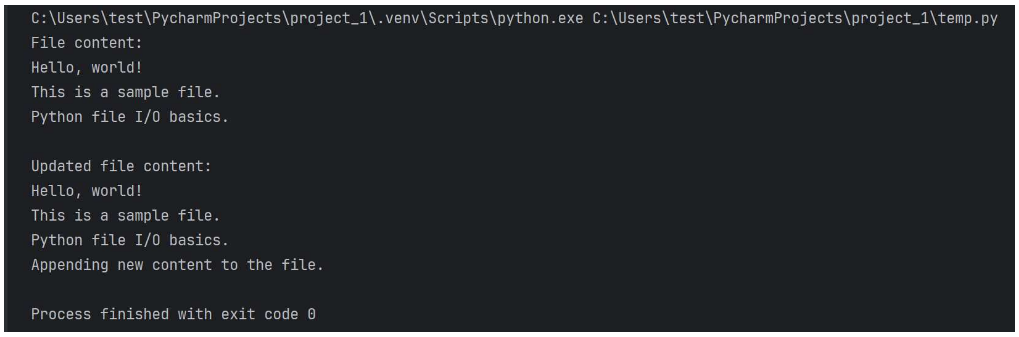
```python
# Writing to a file
with open("sample.txt", "w") as file:
    file.write("Hello, world!\n")
    file.write("This is a sample file.\n")
    file.write("Python file I/O basics.")

# Reading from a file
with open("sample.txt", "r") as file:
    content = file.read()
    print("File content:")
    print(content)

# Appending to a file
with open("sample.txt", "a") as file:
    file.write("\nAppending new content to the file.")

# Reading again to see the changes
with open("sample.txt", "r") as file:
    content = file.read()
    print("\nUpdated file content:")
    print(content)
```

Output:



```
C:\Users\test\PycharmProjects\project_1\.venv\Scripts\python.exe C:\Users\test\PycharmProjects\project_1\temp.py
File content:
Hello, world!
This is a sample file.
Python file I/O basics.

Updated file content:
Hello, world!
This is a sample file.
Python file I/O basics.
Appending new content to the file.

Process finished with exit code 0
```