Working with files in Python

1. **Reading and Writing Files**

- Python provides simple methods to read from and write to files using the built-in `open()` function.
- You can open files in different modes like read (`r`), write (`w`), and append (`a`).

Reading a File:

- Use `open()` with mode `'r'` to read from a file.
- Use `.read()` to read the entire file or `.readlines()` to read the file line by line.

Example of reading a file:

```
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
```

Writing to a File:

- Use `open()` with mode `'w'` to write to a file. This will overwrite the file if it exists.
- Use `'a'` to append content to an existing file.

Example of writing to a file:

```
with open('example.txt', 'w') as file:
    file.write("Hello, this is a test file.\n")
    file.write("Python makes file handling easy.")
```

2. **Handling File Exceptions**

- You should always handle potential errors when working with files, such as file not found or permission issues.
- You can use a `try` and `except` block to handle these exceptions.

Example:

```
try:
    with open('nonexistent_file.txt', 'r') as file:
        content = file.read()
except FileNotFoundError:
    print("The file was not found.")
```

3. **Reading User Input**

- You can use the `input()` function to capture user input.
- By default, `input()` returns a string, so you may need to convert it to another data type (e.g., `int()` or `float()`).

Example:

```
name = input("Enter your name: ")
age = int(input("Enter your age: "))
print(f"Hello {name}, you are {age} years old.")
```

4. **Writing User Input to a File**

- Combine reading user input with file writing. Ask the user for input and save their response to a file.

Example:

```
user_input = input("Enter some text to save to the file: ")
with open('user_input.txt', 'w') as file:
    file.write(user_input)
```