I
E & C
T
IIT Guwahati

Advanced Certification Programme in Data Science Business Analytics

Week 14

Introduction to File Handling (Quick Recap)

Topics Covered

- Functions in Python
- Conditional Statements
- Try-Except Block
- Async Recap
- Q and A

Async Recap

1. Introduction to File Handling:
We explored how programs interact with files to read, write, append, and manage data for real-world applications like reports and logs.

2. Opening and Reading Files in Python:
We learned to use open() with different modes to read files, including how to handle errors like FileNotFoundError and display file content.

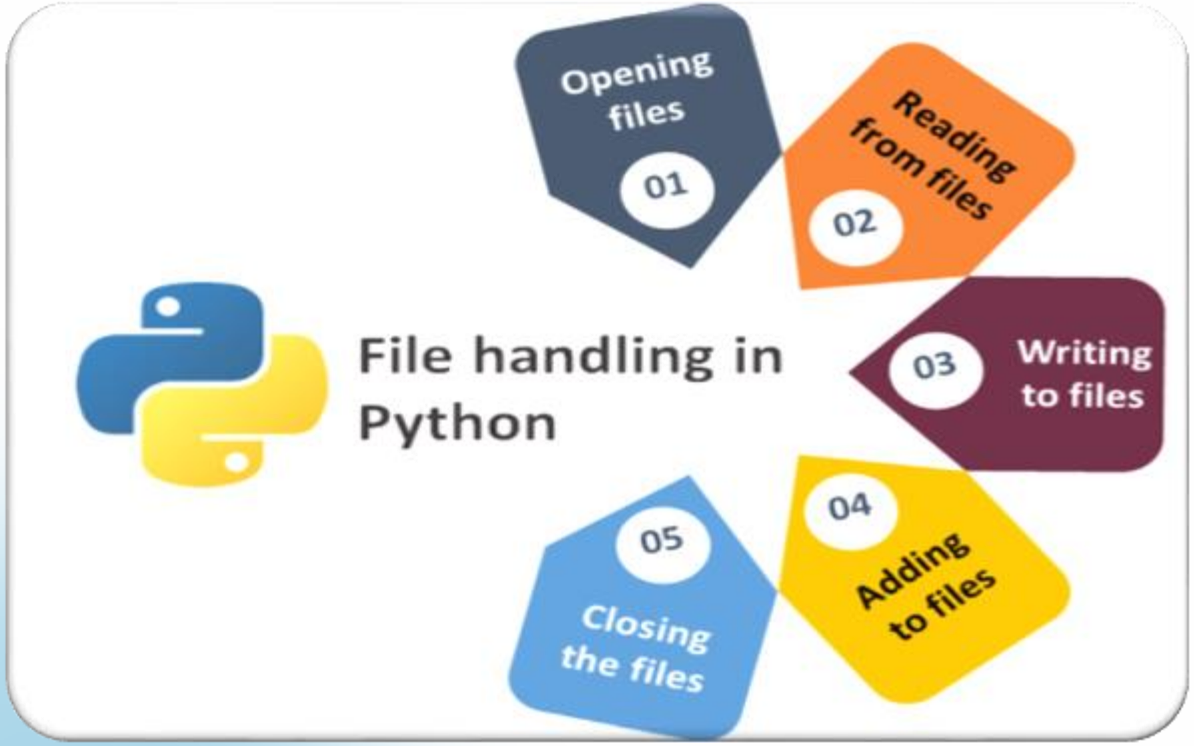
3. Writing, Appending, and Closing Files:
Covered the use of write and append modes to store and update file data, along with the importance of closing files to free system resources.

4. Working with Binary Files:
Understood how to open and read binary files like images using "rb" mode, and safely manage them with with open() for accurate handling.

5. Functions and Error Handling in Python:
Discussed defining functions using def, using return vs print, flexible argument handling with *args and **kwargs, and managing errors with try-except-finally.

File Handling in Python

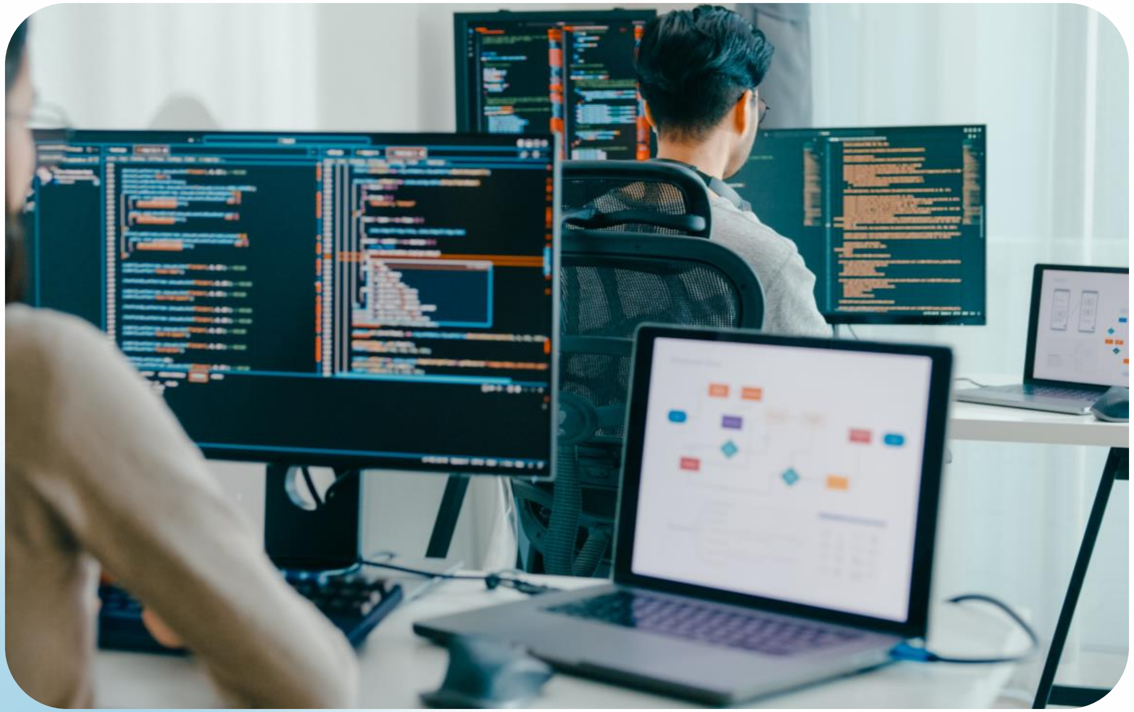
Manage Data with Read, Write and Append Operations



- Enables reading and writing to files
- Supports appending to existing files
- Ensures persistent data management

Advantages of File Handling in Python

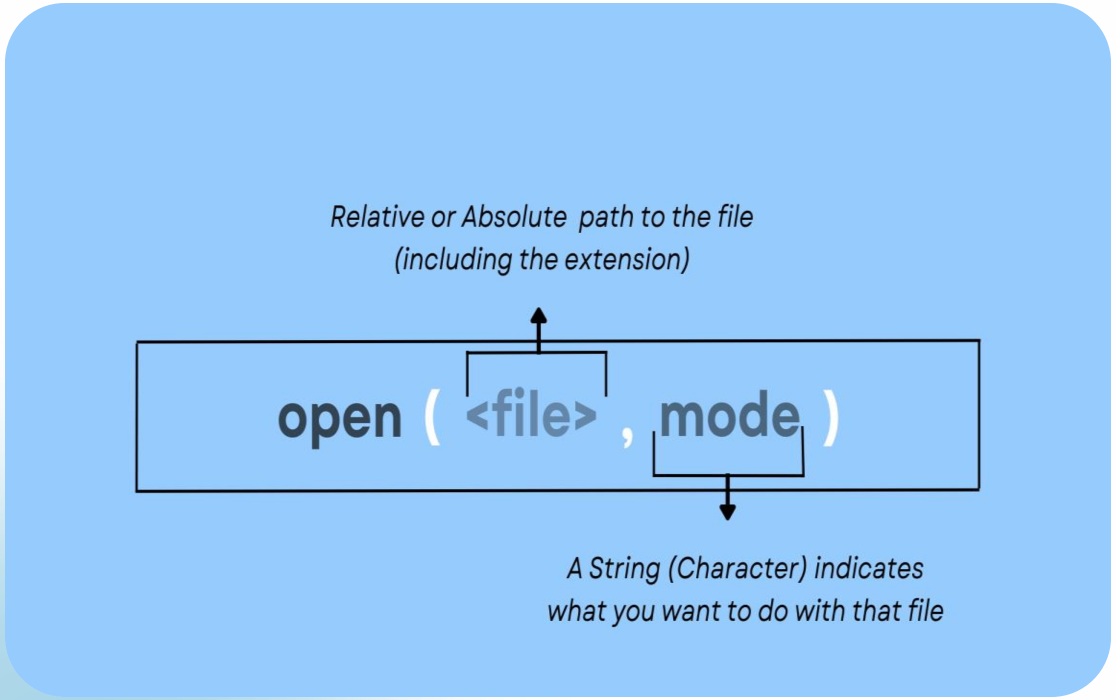
A Powerful, Flexible and User-Friendly Solution



- **Versatility:** Supports create, read, write, append, rename and delete
- **Flexibility:** Supports different operations such as read, write and append
- **User-friendly:** Offers an easy interface to create, read and manipulate files
- **Cross-platform:** Works across Windows, Mac and Linux with seamless integration

Opening a File

Access Files Using Path and Mode



Relative or Absolute path to the file (including the extension)

open (<file> , mode)

A String (Character) indicates what you want to do with that file

- The open() function is used to open a file

- **Syntax:**

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

Modes of file handling in Python

Different Modes for Managing File Operations

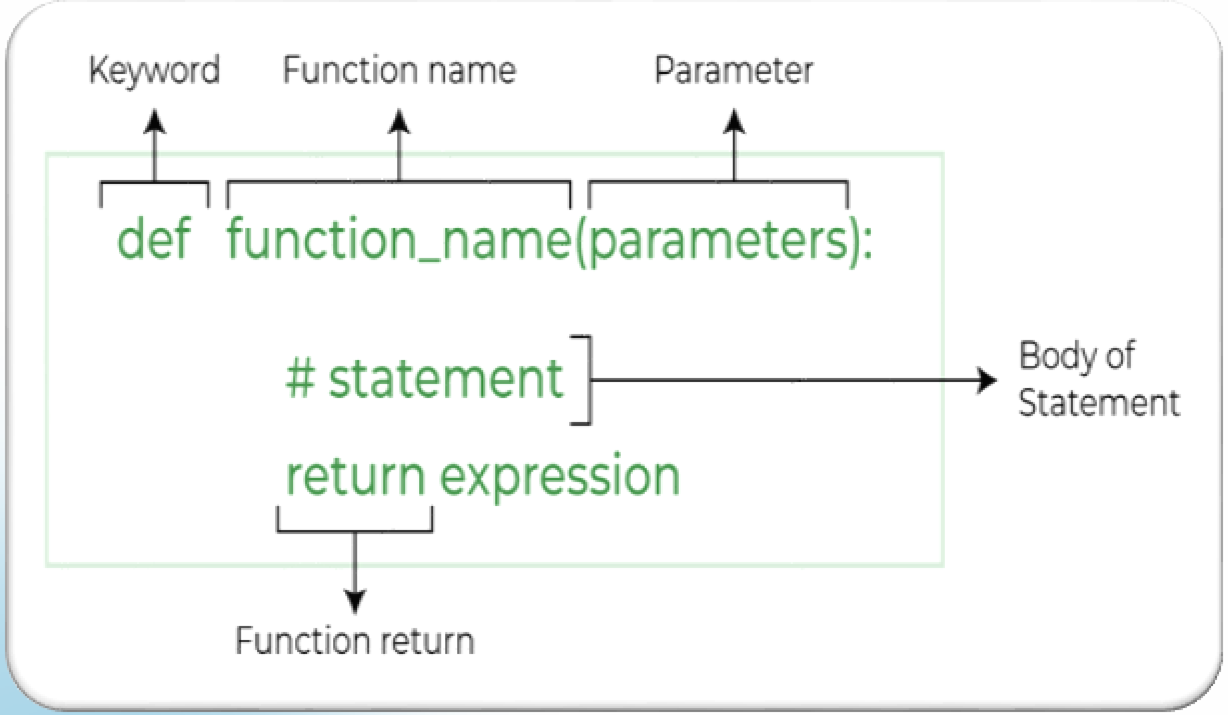
- Python provides various modes using which we can handle files

Mode	Usage
r	Read mode. Opens a file for reading. The file must already exist.
w	Write mode. Opens a file for writing. Creates a new file if it doesn’t exist or overwrites it if it does.
a	Append mode. Opens a file for appending. Creates a new file if it doesn’t exist.
b	Binary mode. Opens a file in binary format.

Functions in Python

What Is a Function?

Improves Modularity and Reuse in Code

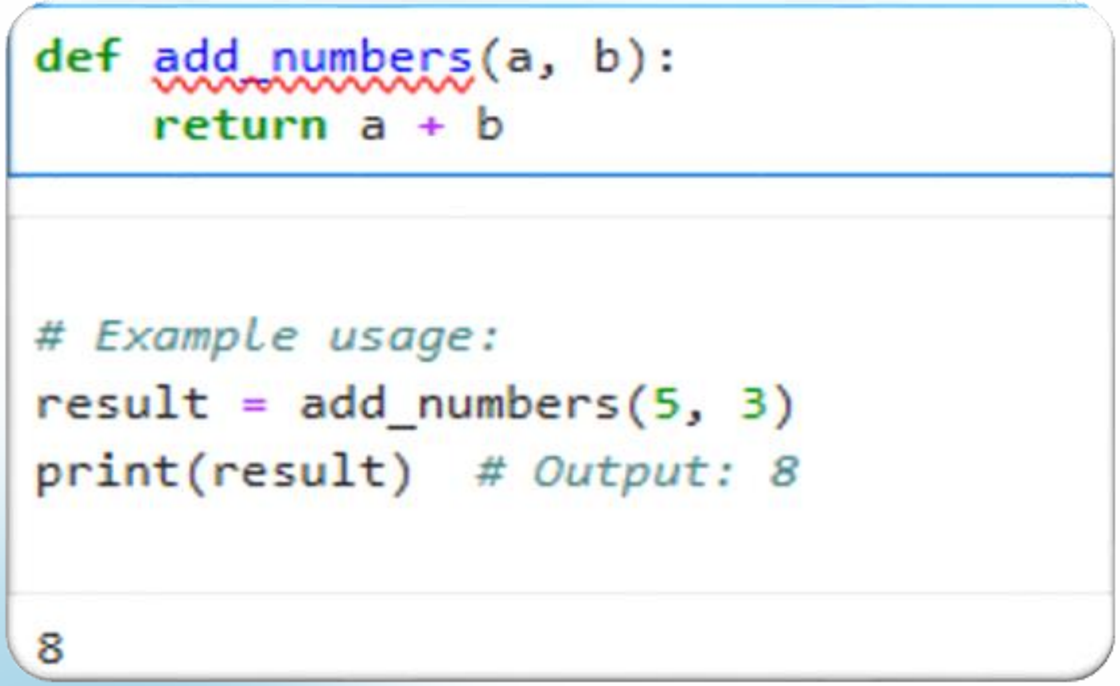


```
def  function_name(parameters):

        # statement

        return expression
```

- A reusable block of code
- Designed to perform a specific task
- Enhances code modularity, readability and reusability

Function to Add Two Numbers

Demonstrates Basic Function with Return Value



```
def add_numbers(a, b):
    return a + b
```

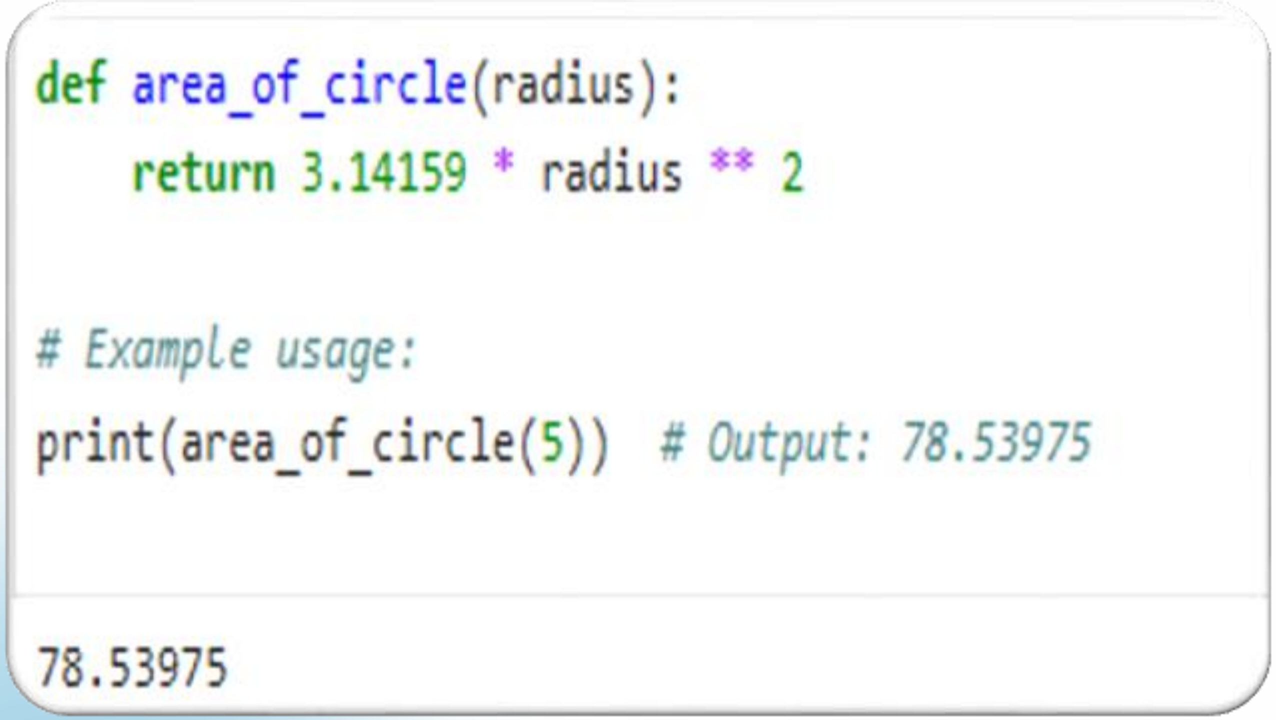
```
# Example usage:
result = add_numbers(5, 3)
print(result)  # Output: 8
```

```
8
```

- Write a function that takes two numbers as arguments and returns their sum

Function to Calculate Area of a Circle

Uses Radius As Input And Returns the Computed Area



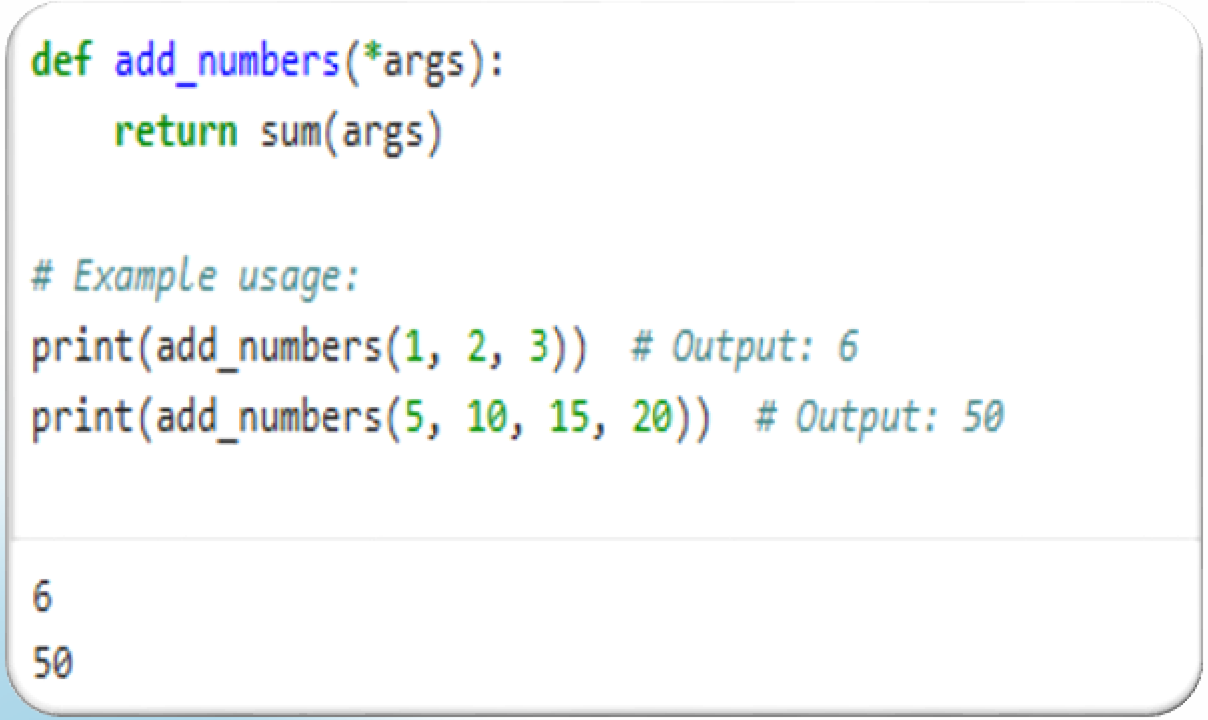
```
def area_of_circle(radius):
    return 3.14159 * radius ** 2


# Example usage:
print(area_of_circle(5))  # Output: 78.53975
```

```
78.53975
```

- Write a function to calculate the area of a circle

Function with Variable Positional Arguments

Adds Multiple Numbers Using *args and sum()



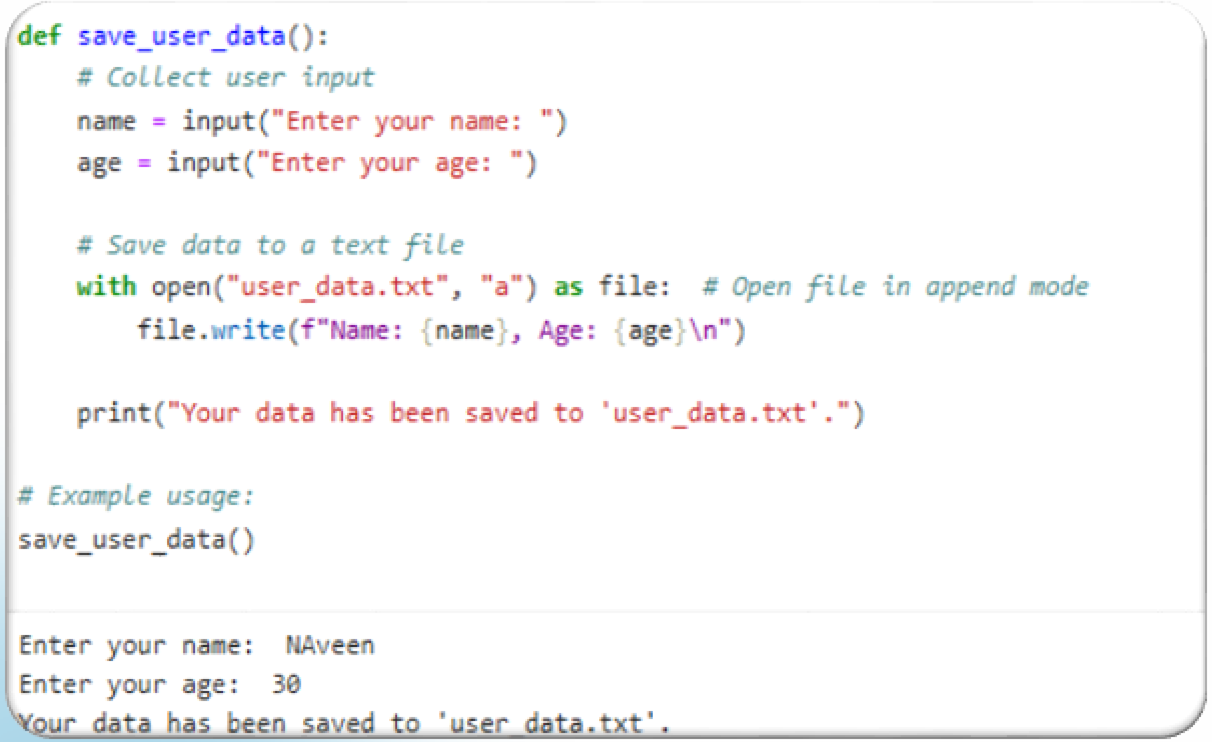
```
def add_numbers(*args):
    return sum(args)

# Example usage:
print(add_numbers(1, 2, 3))  # Output: 6
print(add_numbers(5, 10, 15, 20))  # Output: 50
```

```
6
50
```

- Write a function to accept a variable number of positional arguments and do total of them

Function to Save User Input to a Text File

Store Entered Details in a Persistent Text File



```
def save_user_data():
    # Collect user input
    name = input("Enter your name: ")
    age = input("Enter your age: ")

    # Save data to a text file
    with open("user_data.txt", "a") as file:  # Open file in append mode
        file.write(f"Name: {name}, Age: {age}\n")

    print("Your data has been saved to 'user_data.txt'.")

# Example usage:
save_user_data()
```

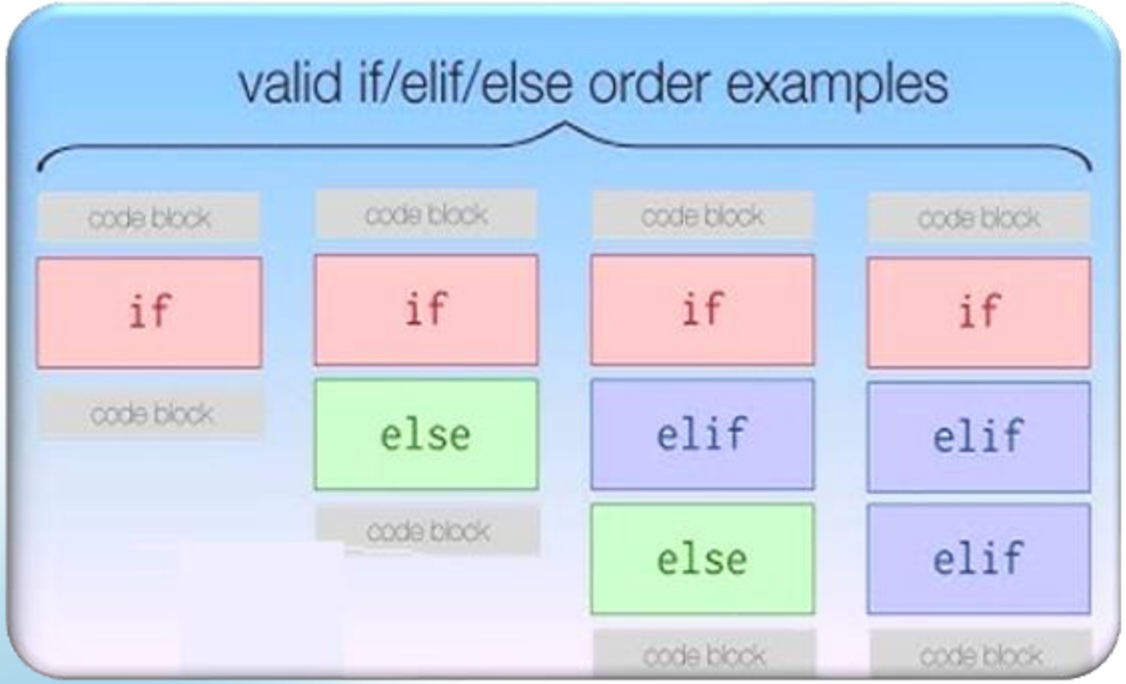
```
Enter your name:  NAveen
Enter your age:  30
Your data has been saved to 'user_data.txt'.
```

- Write a function that can save the data entered by user in a text file

Conditional Statements

Types of Conditional Statement

Direct the Flow of Logic with Structured Conditions



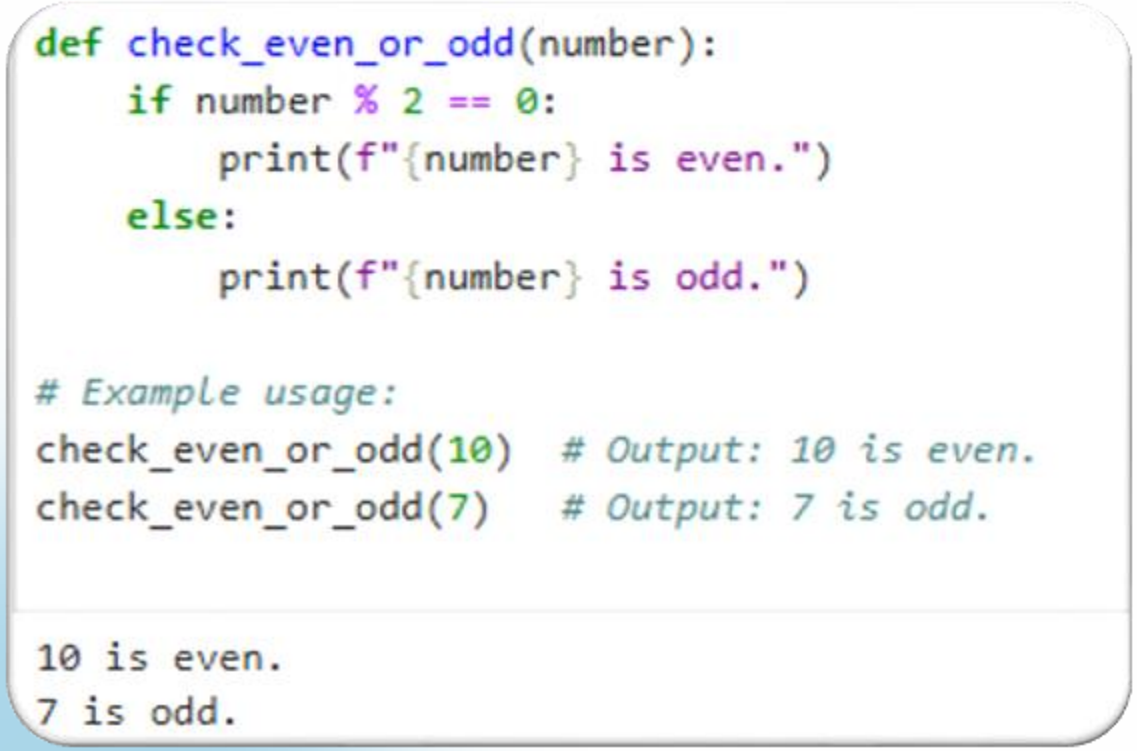
- Conditional statements control program flow

- Conditions are expressions that return True or False

- Common types include if, elif and else

Function to Check Even or Odd

Identifies Whether a Number is Even or Odd



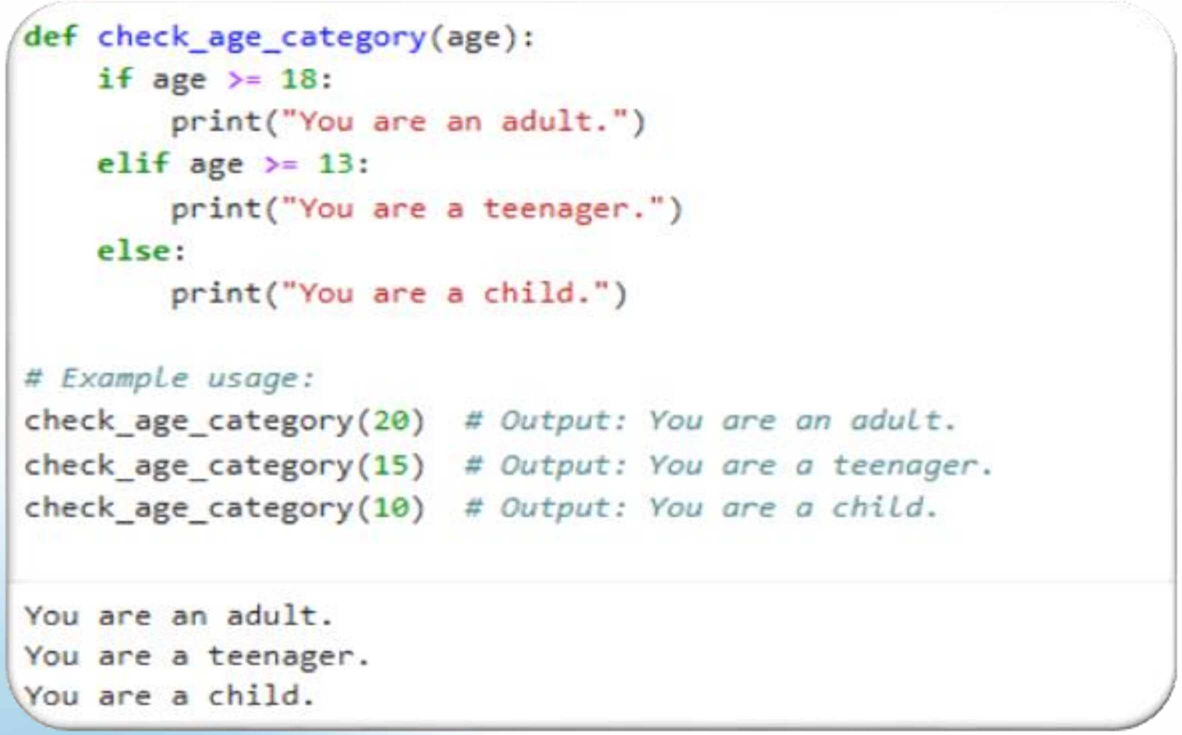
```python
def check_even_or_odd(number):
    if number % 2 == 0:
        print(f"{number} is even.")
    else:
        print(f"{number} is odd.")

# Example usage:
check_even_or_odd(10)  # Output: 10 is even.
check_even_or_odd(7)   # Output: 7 is odd.
```

```
10 is even.
7 is odd.
```

- Write a function that can check whether the number is even or odd

Function to Check Age Category

Classifies Input as Adult, Teenager or Child



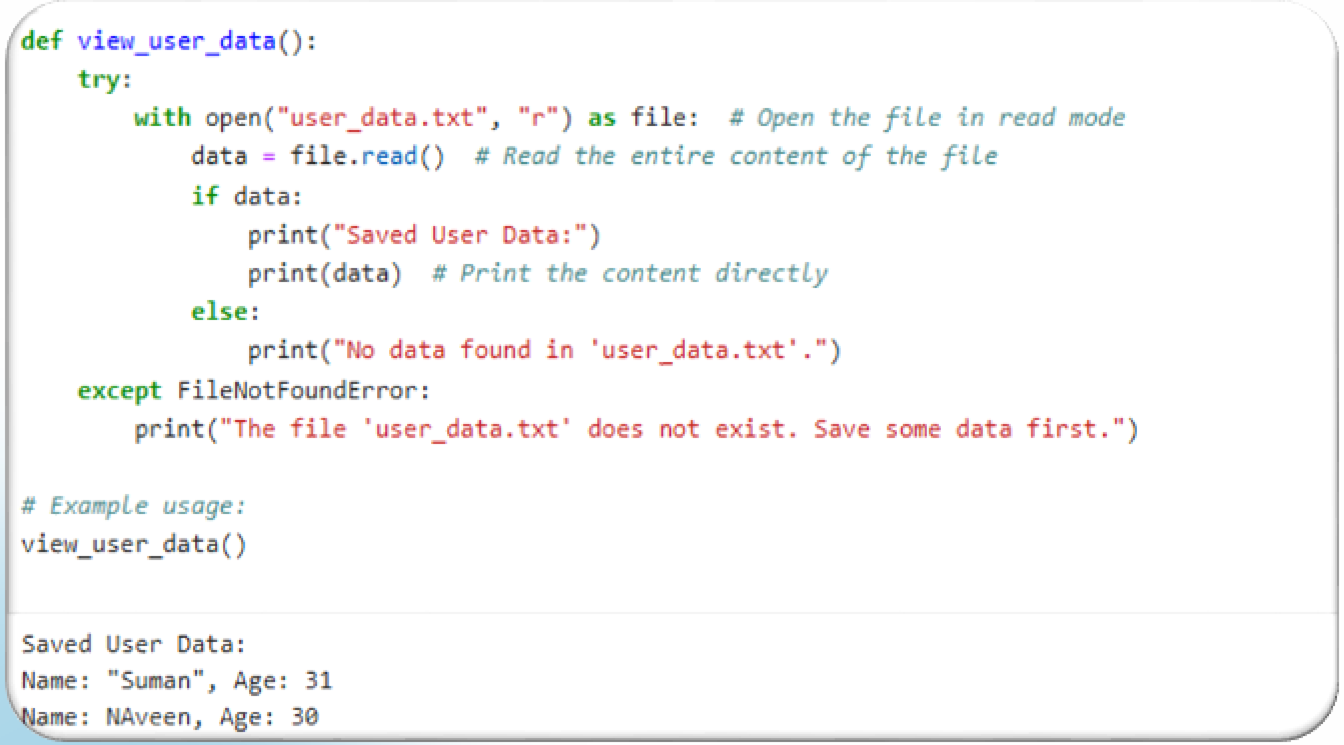
```python
def check_age_category(age):
    if age >= 18:
        print("You are an adult.")
    elif age >= 13:
        print("You are a teenager.")
    else:
        print("You are a child.")

# Example usage:
check_age_category(20)  # Output: You are an adult.
check_age_category(15)  # Output: You are a teenager.
check_age_category(10)  # Output: You are a child.
```

```
You are an adult.
You are a teenager.
You are a child.
```

- Write a function to check whether the person is adult, teenager, or child

Function to Print Saved Data from a Text File

Reads and Displays Content from a Stored File



```python
def view_user_data():
    try:
        with open("user_data.txt", "r") as file:  # Open the file in read mode
            data = file.read()  # Read the entire content of the file
            if data:
                print("Saved User Data:")
                print(data)  # Print the content directly
            else:
                print("No data found in 'user_data.txt'.")
    except FileNotFoundError:
        print("The file 'user_data.txt' does not exist. Save some data first.")

# Example usage:
view_user_data()
```

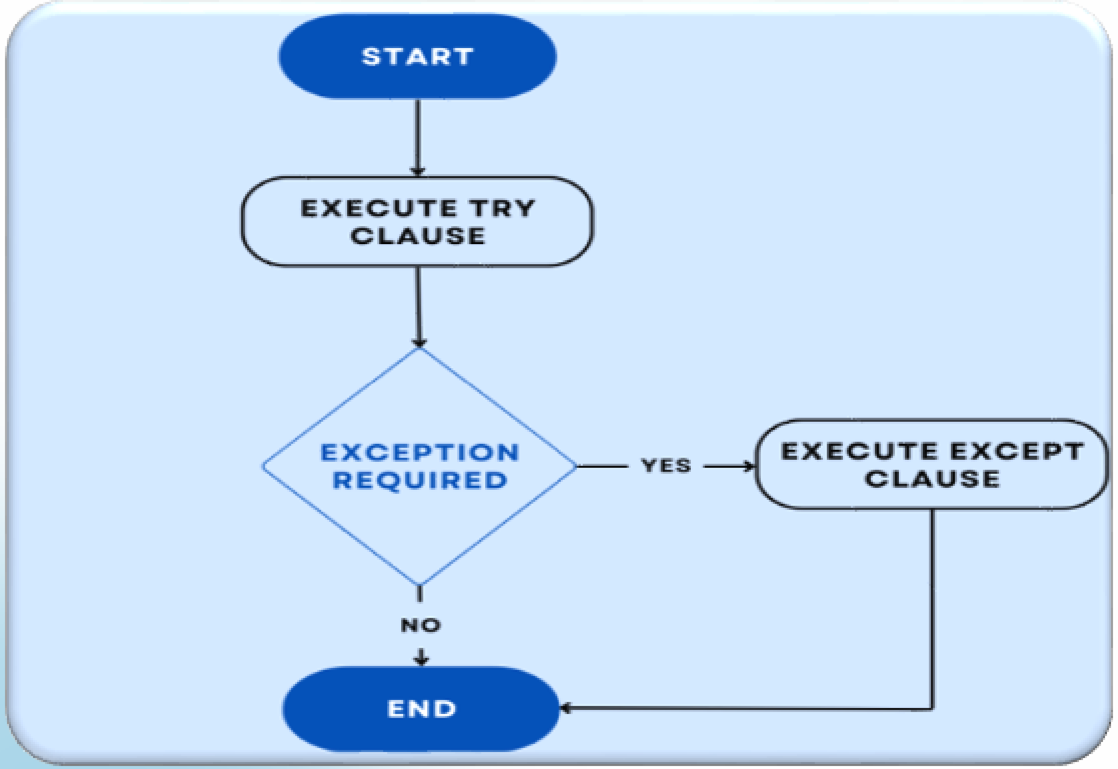
```
Saved User Data:
Name: "Suman", Age: 31
Name: NAveen, Age: 30
```

- Write a function that can print the saved data in a text file

Try-Except Block

What Is Try–Except Statement?

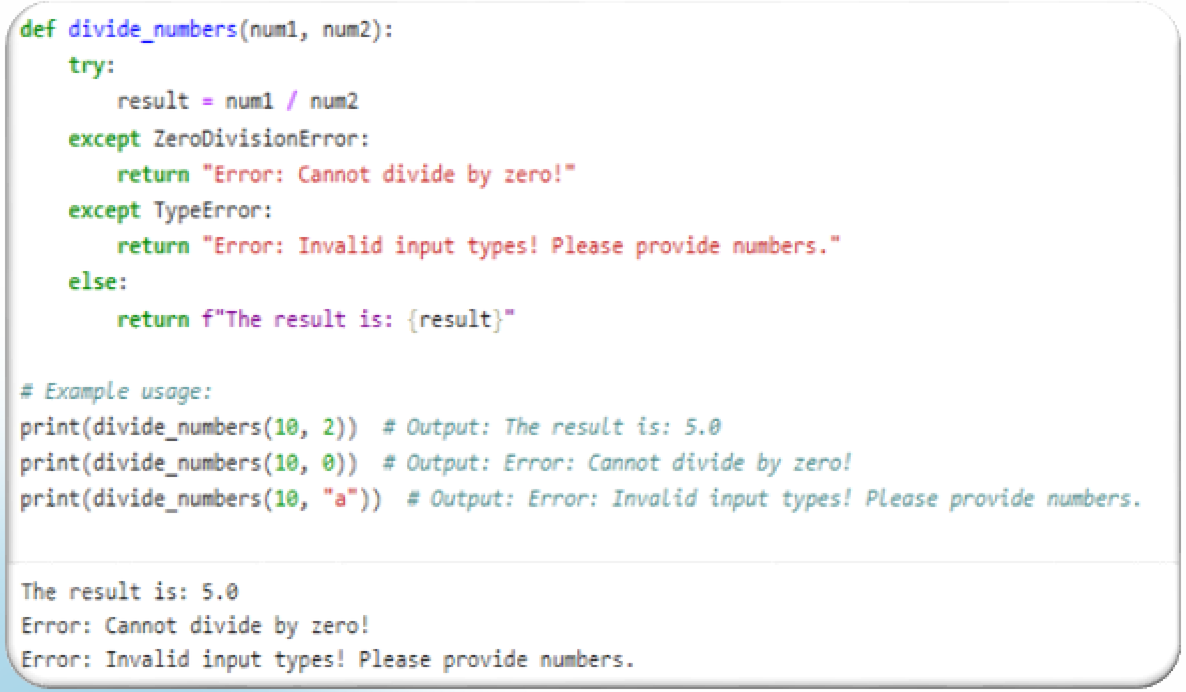
Used To Handle Errors During Program Execution



- Use a try-except statement, to handle errors in a program

Function to Divide Numbers with Error Handling

Handles Division and Detects Invalid Inputs or Zero Division



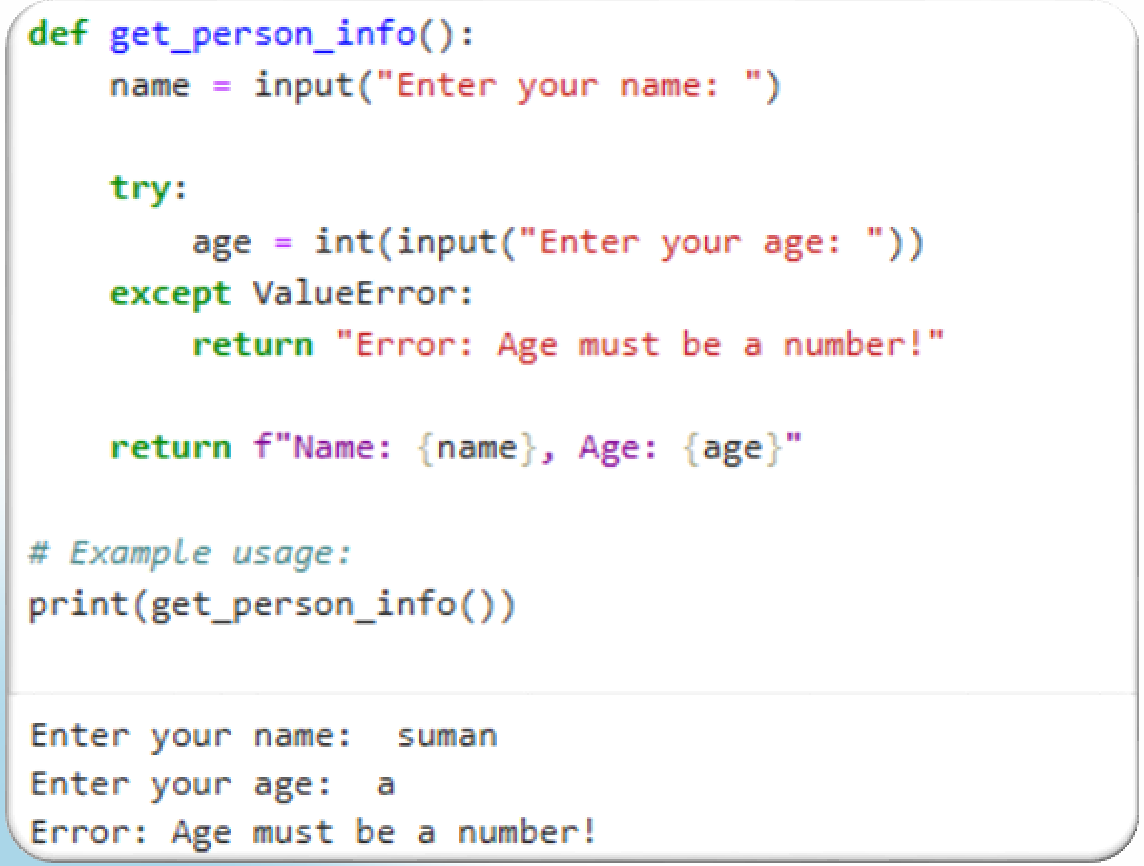
```python
def divide_numbers(num1, num2):
    try:
        result = num1 / num2
    except ZeroDivisionError:
        return "Error: Cannot divide by zero!"
    except TypeError:
        return "Error: Invalid input types! Please provide numbers."
    else:
        return f"The result is: {result}"

# Example usage:
print(divide_numbers(10, 2))  # Output: The result is: 5.0
print(divide_numbers(10, 0))  # Output: Error: Cannot divide by zero!
print(divide_numbers(10, "a"))  # Output: Error: Invalid input types! Please provide numbers.
```

```
The result is: 5.0
Error: Cannot divide by zero!
Error: Invalid input types! Please provide numbers.
```

- Write a function to divide two numbers and handle errors like zero division or invalid input

Handling Errors Gracefully in Python

Create a Function to Divide Numbers and Manage Invalid Inputs



```python
def get_person_info():
    name = input("Enter your name: ")

    try:
        age = int(input("Enter your age: "))
    except ValueError:
        return "Error: Age must be a number!"

    return f"Name: {name}, Age: {age}"

# Example usage:
print(get_person_info())
```

```
Enter your name:  suman
Enter your age:  a
Error: Age must be a number!
```

- Create a function to divide two numbers and handle errors like zero division or invalid input

Q & A

Thank you