CS-121:
Unit 2: Intro. File I/O and Dictionaries

📅 Upcoming

- Today: Introduction to File I/O and Dictionaries
- Tomorrow: Lab: File I/O and Dictionaries
- Thursday: OmniCo Project 1 Due, start Project 2
- This week: Mid-Term Grades

Only 6.5 Weeks Left!

🚀 Final Projects

- Starts in 3 weeks!
- Choose your own adventure
- Work in pairs or solo
- Proposal due in ~2 weeks
- Worth 5x a regular project

Office Hours

- **Today**: 4:30-5:30 PM (200-D)
- **Tomorrow**: 2:30-3:30 PM (During Lab)
- **Thursday**: 4:30-5:30 PM (200-D)

Dictionaries in Python

What is a Dictionary?

- A collection of **key-value pairs**
- Keys are **unique and immutable** (cannot be changed)
- Values can be of any data type (int, str, list, dictionaries, etc.)
- AKA **associative arrays** or **hash maps**

Why Use Dictionaries?

- **Fast data retrieval** using keys
- Organize data with **meaningful labels**
- Ideal for datasets accessed by **unique identifiers**
- **Flexible**: values can be lists, other dictionaries, etc.

Creating a Dictionary

```
# Empty dictionary
my_dict = {}

# Dictionary with initial values
student_grades = {'Alice': 90, 'Bob': 85, 'Charlie': 95}
```

Accessing and Modifying Data

```
# Access value by key
print(student_grades['Alice'])  # Output: 90

# Add or update entries
student_grades['David'] = 88          # Add new entry
student_grades['Alice'] = 92          # Update existing entry

# Remove entries
del student_grades['Bob']
```

Useful Dictionary Methods

```
# Get all keys
keys = student_grades.keys()

# Get all values
values = student_grades.values()

# Get all key-value pairs
items = student_grades.items()
```

Iterating Over a Dictionary

```
# Loop through keys
for student in student_grades:
    print(student)

# Loop through keys and values
for student, grade in student_grades.items():
    print(f"{student}: {grade}")
```

Real-World Example: Word Counting

```
word_counts = {}
for word in text.split():
    word = word.lower()
    if word in word_counts:
        word_counts[word] += 1  # Increment count
    else:
        word_counts[word] = 1   # Initialize count
```

- Useful for **text analysis**, **data mining**, etc.

📂 File I/O in Python

What is File I/O?

- **Input/Output operations** with files
- **Reading** data from files
- **Writing** data to files
- Essential for **data persistence** and handling external data

Why Use File I/O?

- **Store data** between program runs
- Process **large datasets**
- Read **configuration files**
- Generate **reports or logs**

Opening Files in Python

```
# Open a file in read mode
file = open('filename.txt', 'r')
```

- Common modes:
  - `'r'`: **Read** (default)
  - `'w'`: **Write** (overwrites)
  - `'a'`: **Append** (adds to end)
  - `'r+'`: **Read and write**

Reading from a File

```
# Read entire content
with open('data.txt', 'r') as file:
    content = file.read()

# Read line by line
with open('data.txt', 'r') as file:
    for line in file:
        print(line.strip())
```

Writing to a File

```
# Write content (overwrites existing file)
with open('output.txt', 'w') as file:
    file.write("Hello, World!")

# Append content to the end of the file
with open('output.txt', 'a') as file:
    file.write("\nAdditional line.")
```

The `with` Statement

- **Automatically** handles file closing
- Ensures **resources** are properly released
- Preferred way to work with files

```
# Without `with`, you need to manually close the file
file = open('filename.txt', 'r')
data = file.read()
file.close()  # Important to close the file

with open('filename.txt', 'r') as file:
    # Work with the file
    data = file.read()
# File is automatically closed here
```

Why Use the `with` Statement?

- If you **forget to close** the file:
  - Can lead to **resource leaks**
  - May run into **errors** when trying to open the file again
  - The `with` statement helps **prevent these issues**

Real-World Example: Reading CSV Files

```
with open('data.csv', 'r') as file:
    for line in file:
        values = line.strip().split(',')
        # Process values
```

- Commonly used for **data exchange** and **storage**

✅ Putting it All Together

Reading a File into a List

```
# Read lines from a file
with open('students.txt', 'r') as file:
    lines = file.readlines()

# Clean up data by stripping whitespace
students = [line.strip() for line in lines]
```

Example: `students.txt` Content

```
Alice,90
Bob,85
Charlie,95
```

Creating a Dictionary from a List

```
# Initialize an empty dictionary
student_grades = {}

# Populate the dictionary
for student in students:
    name, grade = student.split(',')
    student_grades[name] = int(grade)
```

Using the Data

```
# Access grades by student name
print(student_grades['Alice'])  # Output: 90

# Calculate average grade
average = sum(student_grades.values()) / len(student_grades)
print(f"Class average: {average}")
```

Real-World Applications

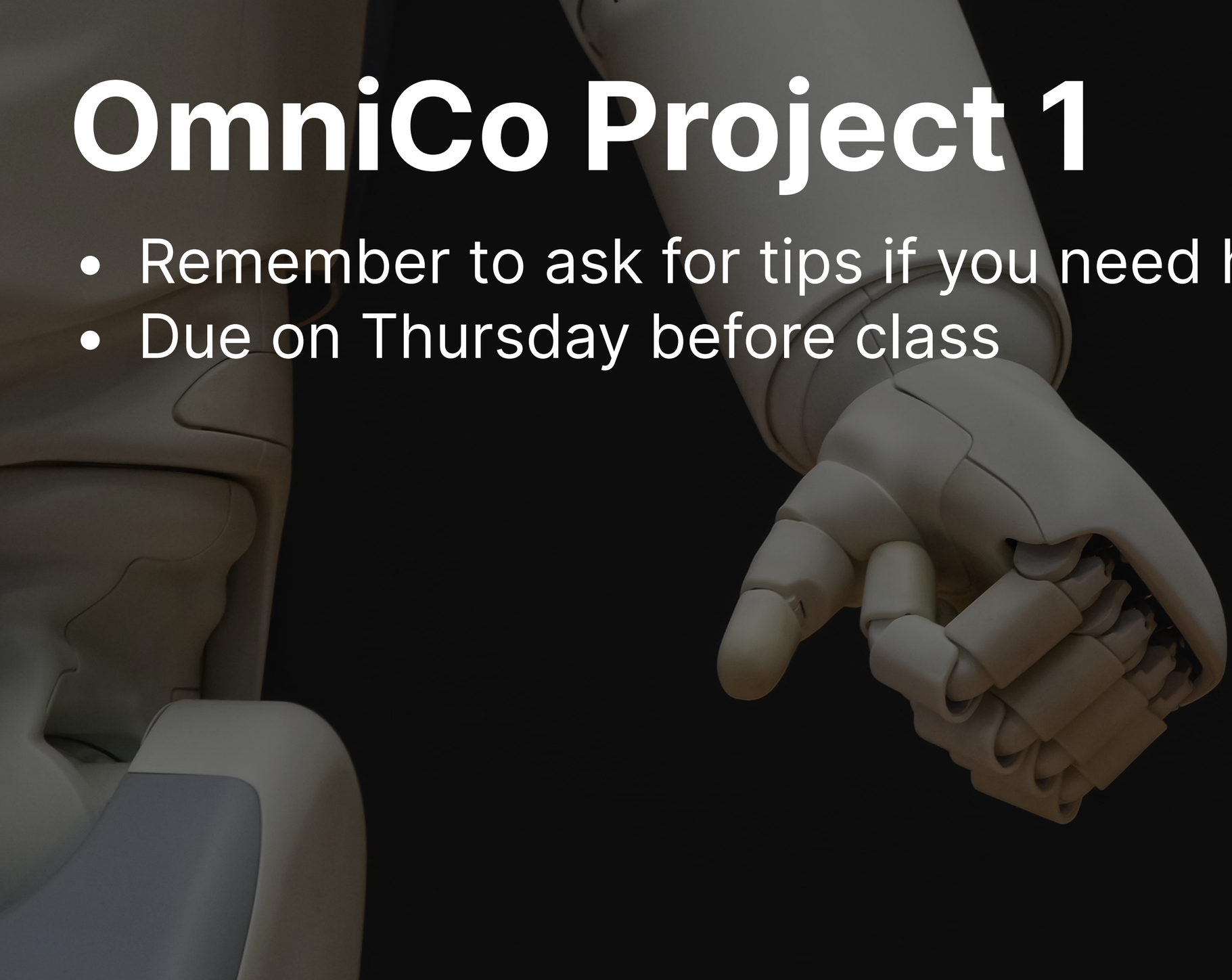
- **Data Analysis**: Read and process datasets
- **User Management**: Load user info from files
- **Configuration**: Parse settings into dictionaries

Questions?

- Any questions about **Dictionaries** or **File I/O**?
- Ready to apply these concepts in **tomorrow's lab**!

OmniCo Project 1

- Remember to ask for tips if you need help
- Due on Thursday before class



OMNICO

Welcome to the future.

Welcome home.

Welcome to OmniCo.

Thank You!

- Don't forget to **start on Project 2**
- See you during **office hours**!