Session 2.2: List Comprehensions & Advanced Operations

Powerful Data Processing: Week 2, Day 2

Today's Journey

- Part 1: List Comprehensions - Power Tool!
- Part 2: Workshop - Transform Data Like a Pro
- Break
- Part 3: Advanced List Operations
- Part 4: Workshop - Complex Data Processing
- Part 5: Quiz & Week 2 Wrap-up

Part 1: List Comprehensions

The most powerful tool in Python!

Week 2, Session 2 - You're Halfway! 🎉

Last session you learned:

- ✓ Lists (collections of data)
- ✓ Indexing and slicing
- ✓ List methods (append, sort, etc.)
- ✓ GitHub (code online!)

Today: Super-powered list operations!

List comprehensions =

- Write less code
- Process data faster
- Look like a professional
- Industry standard in Python

What Are List Comprehensions? (Simple Answer)

Comprehension = create new list from existing data

Think of it as:

- Excel formula for entire column (apply to all)
- Recipe that transforms ingredients
- Assembly line that modifies each item

One line instead of 5 lines!

Before you learn comprehensions:

- 5 lines of code with loop

After you learn comprehensions:

- 1 line of elegant code

Same result, professional style! 🚀

Loop vs Comprehension - Side by Side

```
# Task: Double each number
numbers = [1, 2, 3, 4, 5]

# OLD WAY: 5 lines with loop
doubled = []
for num in numbers:
    doubled.append(num * 2)
print(doubled)  # [2, 4, 6, 8, 10]

# NEW WAY: 1 line with comprehension
doubled = [num * 2 for num in numbers]
print(doubled)  # [2, 4, 6, 8, 10]

# SAME RESULT!
# Which would you rather write?
```

List Comprehension Syntax (Memorize This!)

Format:

- new_list = [EXPRESSION for ITEM in OLD_LIST]

Read it like English:

- 'Make a list of EXPRESSION for each ITEM in OLD_LIST'

  Example: [x*2 for x in numbers]

- 'Make a list of x*2 for each x in numbers'

Parts:

- [ ] - creates list
- EXPRESSION - what to do (x*2, x.upper(), etc.)
- for ITEM - loop variable
- in OLD_LIST - source data

Once you see the pattern, it's easy! 💡

Your First Comprehensions - Practice!

```
# Square each number (perfect!)
nums = [1, 2, 3, 4, 5]
squares = [n**2 for n in nums]
print(squares)  # [1, 4, 9, 16, 25]

# Make all uppercase (spot on!)
names = ['alice', 'bob', 'carol']
upper = [name.upper() for name in names]
print(upper)  # ['ALICE', 'BOB', 'CAROL']

# Capitalize first letter (great use of .capitalize()!)
names = ['alice', 'bob', 'carol', 'carlos']
capitalized_names = [name.capitalize() for name in names]
print(capitalized_names)  # ['Alice', 'Bob', 'Carol', 'Carlos']

# Get string lengths (correct now!)
lengths = [len(name) for name in names]
print(lengths)  # [5, 3, 5, 6]

words = ['hi', 'hello', 'hey', 'goodbye']
lengths = [len(word) for word in words]
print(lengths)  # [2, 5, 3, 7]
```

Why Comprehensions Matter for ML

Machine Learning = transforming LOTS of data

Real ML tasks using comprehensions:

- Normalize features: [x/max_val for x in features]
- Scale to 0-1: [(x-min)/(max-min) for x in data]
- Convert labels: [1 if x=='yes' else 0 for x in labels]
- Extract columns: [row[3] for row in dataset]

You'll write these EVERY DAY in ML!

Professional data scientists use comprehensions

It's the Pythonic way! ✨

Adding Filters with 'if' (Game Changer!)

Format:

- new_list = [EXPRESSION for ITEM in LIST if CONDITION]

The 'if' at the end filters items!

Example: [x for x in numbers if x > 5]

- 'Make list of x for each x in numbers IF x > 5'

Two powers combined:

1. Transform data (EXPRESSION)
2. Filter data (if CONDITION)

This is incredibly powerful!

One line does both! 🔥

Comprehensions with Filtering

```
# Get only even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = [n for n in numbers if n % 2 == 0]
print(evens)  # [2, 4, 6, 8, 10]

# Get only positive numbers
values = [-5, 3, -1, 8, -2, 10, 0]
positive = [v for v in values if v > 0]
print(positive)  # [3, 8, 10]

# Get names longer than 4 letters
names = ['Jo', 'Alice', 'Bob', 'Carol', 'Dave']
long_names = [name for name in names if len(name) > 4]
print(long_names)  # ['Alice', 'Carol']
# Filter unwanted data!
```

Transform AND Filter Together!

```
# Square only the even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
even_squares = [n**2 for n in numbers if n % 2 == 0]
print(even_squares)  # [4, 16, 36, 64]

# Uppercase only long names
names = ['Jo', 'Alice', 'Bob', 'Carol']
upper_long = [n.upper() for n in names if len(n) > 3]
print(upper_long)  # ['ALICE', 'CAROL']

# Double only numbers above 50
scores = [45, 67, 52, 38, 91, 73]
doubled_high = [s*2 for s in scores if s > 50]
print(doubled_high)  # [134, 104, 182, 146]
# This is data science magic! ✨
```

Real-World Example: Data Cleaning

Imagine: Temperature sensor sometimes gives errors

Error value: -999 (impossible temperature)

Your job:

1. Remove errors
2. Convert Fahrenheit → Celsius
3. Round to 1 decimal place

With comprehensions: Easy!

Without comprehensions: Lots of code

This is real data science work!

Companies pay for this skill! 💰

Data Cleaning with Comprehensions

```
# Raw sensor data (Fahrenheit)
temps_f = [98.6, 105.2, -999, 72.1, -999, 101.3, 98.9]

# Clean and convert in ONE line!
# Remove -999, convert to Celsius, round
temps_c = [
    round((temp - 32) * 5/9, 1)
    for temp in temps_f
    if temp > 0  # Filter out errors
]
print(temps_c)

# [37.0, 40.7, 22.3, 38.5, 37.2]
# Clean data in one comprehension!
# This is professional data cleaning! 🧹
```

More Real Examples You'll Use

```
# Extract specific column from data
students = [
    ['Alice', 95, 'A'],
    ['Bob', 87, 'B'],
    ['Carol', 92, 'A']
]
names = [student[0] for student in students]

# ['Alice', 'Bob', 'Carol']
# Convert text labels to numbers (ML!)
labels = ['yes', 'no', 'yes', 'yes', 'no']
binary = [1 if label == 'yes' else 0 for label in labels]
# [1, 0, 1, 1, 0]
# This is feature engineering!
# Core ML skill!
```

Using range() with Comprehensions

Combine range() with comprehensions!

Generate data instantly

Common patterns:

- First N squares: [i**2 for i in range(N)]
- Even numbers: [i for i in range(20) if i%2==0]
- Specific pattern: [i*5 for i in range(10)]

Super fast data generation!

Perfect for testing code

Create sample datasets instantly

range() + Comprehensions = Power!

```
# First 10 squares
squares = [i**2 for i in range(10)]
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# Even numbers 0 to 20
evens = [i for i in range(21) if i % 2 == 0]
# [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

# Multiples of 5 up to 50
mult_5 = [i for i in range(0, 51, 5)]
# [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

# Generate test data instantly!
test_data = [i * 0.1 for i in range(100)]
# [0.0, 0.1, 0.2, ..., 9.9]
```

When to Use Comprehensions vs Loops

✅ USE Comprehensions when:

- Transforming data (multiply, convert, format)
- Filtering data (only certain values)
- Creating new list from old list
- Simple, readable one-liner

✅ USE Regular Loops when:

- Complex logic (multiple conditions)
- Multiple operations per item
- Modifying original list
- Debugging complex code

Both are valid!

Start with what's clearer to you

Comprehensions come with practice 🎯

Part 2: Workshop

Master Comprehensions!

💻 Exercise 1: Comprehension Basics

Create file: comprehension_practice.py

Given: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Create using comprehensions:

1. List of all numbers squared
2. List of only odd numbers
3. List of numbers divisible by 3
4. List of numbers squared, but only if even
5. List containing 'even' or 'odd' for each number

Print each result with descriptive label

Example output:

- Squares: [1, 4, 9, 16, ...]
- Odds: [1, 3, 5, 7, 9]

💻 Exercise 2: Temperature Converter

Create file: temp_converter.py

Given temps in Fahrenheit:

- temps_f = [32, 68, 98.6, 212, 72, 86, 50, 95]

Tasks using comprehensions:

1. Convert all to Celsius: (F - 32) * 5/9
2. Round each to 1 decimal place
3. Filter: only temps above 20°C
4. Create labels: 'Hot' if >= 25°C, else 'Cold'
5. Print original F and converted C side-by-side

Use f-strings for professional output!

This is real data conversion! 🌡️

💻 Exercise 3: Data Cleaning Challenge

Create file: data_cleaner.py

Given messy sensor data:

- data = [10, -999, 25, 30, -999, 15, 40, -999, 35, 20]

(-999 means sensor error - remove it!)

Tasks with comprehensions:

1. Remove all -999 values (filter)
2. Calculate average of clean data
3. Find all values above average
4. Normalize to 0-1 range:
   - normalized = (value - min) / (max - min)
5. Print before/after comparison

This is real data cleaning!

You're doing data science! 🖌️

Great Work!

Comprehensions are powerful!

Part 3: Advanced List Operations

Level up your list skills!

Nested Lists - Data in 2D

Nested list = list of lists

Like a table or spreadsheet

Real-world examples:

- Gradebook (students × subjects)
- Image pixels (rows × columns)
- Game board (grid)
- Dataset (samples × features)

Format: [[row1], [row2], [row3]]

Access: list[row][column]

Foundation for NumPy arrays! 📊

Creating and Using Nested Lists

```
# Student grades: [student][subject]
# Columns: Math, English, Science
grades = [
    [95, 87, 92],  # Alice
    [88, 90, 85],  # Bob
    [92, 88, 94],  # Carol
    [78, 85, 80]   # David
]

# Access specific grade
print(grades[0][0])  # 95 (Alice's Math)
print(grades[1][2])  # 85 (Bob's Science)

# Get all of Alice's grades
print(grades[0])  # [95, 87, 92]
# This is like Excel!
# Row 1, Column 1 = grades[0][0]
```

Processing Nested Lists

```
grades = [
    [95, 87, 92],
    [88, 90, 85],
    [92, 88, 94],
    [78, 85, 80]
]

# Calculate average for each student
for student_grades in grades:
    avg = sum(student_grades) / len(student_grades)
    print(f'Average: {avg:.1f}')

# Output:
# Average: 91.3
# Average: 87.7
# Average: 91.3
# Average: 81.0
# Or with comprehension!

avgs = [sum(row)/len(row) for row in grades]
```

String split() and join() - Essential!

Two powerful string methods for lists:

- split() - String → List
  - Breaks string into pieces
  - Default: split on spaces
  - Returns list of strings
- join() - List → String
  - Combines list into string
  - Specify separator
  - Returns single string

Used constantly in text processing!

Natural Language Processing (NLP) basics 

split() - String to List

```
# Split sentence into words
sentence = 'I love learning Python for ML'
words = sentence.split()
print(words)

# ['I', 'love', 'learning', 'Python', 'for', 'ML']
# Count words
word_count = len(words)
print(f'Words: {word_count}')  # 6

# Split on different character
csv_line = 'Alice,25,Engineer,Boston'
fields = csv_line.split(',')
print(fields)
# ['Alice', '25', 'Engineer', 'Boston']
# This is how CSV files work!
```

join() - List to String

```
# Join words with spaces
words = ['I', 'love', 'Python']
sentence = ' '.join(words)
print(sentence)  # 'I love Python'

# Join with different separator
words = ['apple', 'banana', 'cherry']
csv = ','.join(words)
print(csv)  # 'apple,banana,cherry'

# Join with newlines
lines = ['Line 1', 'Line 2', 'Line 3']
text = '\n'.join(lines)
print(text)
# Line 1
# Line 2
# Line 3
# Essential for file writing!
```

List Copying - CRITICAL Warning! ⚠️

Lists are mutable (changeable)

Assignment creates REFERENCE, not copy!

The trap:

- list2 = list1 # Both point to SAME list!
- list2.append(4) # Changes BOTH lists!

This catches EVERYONE at first!

Solution: Use .copy()

- list2 = list1.copy() # Real copy

Now they're independent!

Changes to one don't affect the other 🎯

List Copying - Right vs Wrong

```
# WRONG! (This is a trap!)
original = [1, 2, 3]
reference = original  # Not a copy!
reference.append(4)
print(original)   # [1, 2, 3, 4] - CHANGED!
print(reference)  # [1, 2, 3, 4]

# RIGHT! Make actual copy
original = [1, 2, 3]
copy = original.copy()  # Real copy
copy.append(4)
print(original)  # [1, 2, 3] - Unchanged!
print(copy)      # [1, 2, 3, 4]

# Other ways to copy:
# copy = list(original)
# copy = original[:]
# All three create real copies!
```

zip() - Combine Two Lists

zip() pairs up items from multiple lists

Like a zipper on clothing:

- Combines two sides
- Creates pairs
- Line by line

Super useful when you have:

- Names and scores
- Dates and values
- Keys and values
- Any parallel data

Common in data science! 🔗

Using zip() to Combine Lists

```
# Two parallel lists
names = ['Alice', 'Bob', 'Carol', 'David']
scores = [95, 87, 92, 78]

# Combine with zip()
for name, score in zip(names, scores):
    print(f'{name}: {score}')
# Output:
# Alice: 95
# Bob: 87
# Carol: 92
# David: 78

# Create dictionary from two lists
grade_dict = dict(zip(names, scores))
print(grade_dict)
# {'Alice': 95, 'Bob': 87, 'Carol': 92, 'David': 78}
```

Common List Patterns (Memorize These!)

Patterns you'll use constantly:

1. Accumulator: Build list in loop
    - results = []
    - for item in data: results.append(process(item))
2. Filter: Keep only matching items
    - filtered = [x for x in data if condition]
3. Transform**: Modify each item
    - transformed = [func(x) for x in data]
4. Aggregate: Combine to single value
    - total = sum(data)
    - average = sum(data) / len(data)

These patterns = 90% of data processing! 📊

Part 4: Workshop

Advanced List Mastery!

💻 Exercise 4: Complete Grade Book

Create file: gradebook_advanced.py

Build professional grade management system:

1. Create nested list of grades:

grades = [[95,87,92,88], [88,90,85,92],

[92,88,94,90], [78,85,80,88]]

students = ['Alice', 'Bob', 'Carol', 'David']

2. For EACH student calculate:

- Average grade
- Highest grade
- Lowest grade
- Number of A's (>= 90)

3. Calculate CLASS statistics:

- Overall class average
- Highest student average
- Lowest student average
- Total A's in class

4. Use comprehensions where possible

5. Format output professionally

This is real gradebook software! 📊

Part 5: Quiz & Wrap-up

Week 2 Complete!

Session 2.2 Quiz

🎉 WEEK 2 COMPLETE! You're Amazing!

This week you mastered:

- ✓ Lists (creation, indexing, slicing, methods)
- ✓ List comprehensions (professional Python!)
- ✓ Nested lists (2D data structures)
- ✓ Advanced operations (zip, split, join)
- ✓ GitHub (code portfolio online!)

You can now:

- Store and process data collections
- Transform data in one line
- Clean messy datasets
- Build data structures
- Share code professionally

This is SERIOUS progress! 🌟

Homework - Build Your Portfolio!

1. Complete the quiz (10 questions - required)
2. Project: Complete Data Analyzer
    - Build: data_analyzer_pro.py
    - Load dataset: [list of 50 numbers]
    - Calculate: mean, median, std deviation
    - Find outliers (>2 std dev from mean)
    - Clean data (remove outliers)
    - Show before/after statistics
    - Use comprehensions throughout!
3. Git/GitHub Practice
    - Commit all Week 2 work
    - Push to GitHub
    - Check your green squares!
    - Add README.md to explain your projects
4. Prepare for Week 3: Read about for loops

Looking Ahead: Week 3 - Automation! 🚀

Next session: For Loops (Automation Begins!)

You'll learn:

- for loops (repeat tasks automatically)
- range() function (counting & iterating)
- enumerate() (track position)
- Loop patterns (building data)
- Git branching (parallel development)

Why loops matter:

- Process thousands of items automatically
- ML training = looping through epochs
- Data processing = loop through datasets
- Automation = programmer's superpower

Difficulty increases slightly - you're ready! 💪

Week 2 Reflection - Amazing Growth! 🌱

Think about your journey:

Monday (Week 2, Day 1):

- 'What's a list?'

Now (Week 2, Day 2):

- 'I write comprehensions and process nested data!'

You learned:

- Data collections (lists)
- Professional Python (comprehensions)
- Online portfolio (GitHub)
- Real data processing

Next week: Loops, automation, branches

You're becoming a real programmer! 🎓

Weekend practice: Review and build projects!

See you Monday for Week 3! 🚀