Session 2.1: Lists - Your First Data Collection

Storing Multiple Values: Week 2, Day 1

Today's Journey

- Part 1: What Are Lists? Collections Explained
- Part 2: Workshop - Build Your First Collections
- Break - Week 2 is Here!
- Part 3: GitHub - Share Your Code with the World
- Part 4: Workshop - Push to GitHub
- Part 5: Quiz & Homework

Part 1: Lists Fundamentals

Collections are the foundation of data science!

Welcome to Week 2! 🎉

Last week you learned:

- ✓ Variables (single values)
- ✓ Strings (text)
- ✓ Git (saving your work)

This week: Work with MANY values at once!

Real world needs:

- List of customer names
- Collection of temperatures
- Set of test scores
- Multiple measurements

Lists solve this problem! 🚀

What Are Lists? (Simple Explanation)

List = collection of values in order

Think of it like:

- Shopping list (groceries)
- To-do list (tasks)
- Playlist (songs)
- Excel column (data)

In Python:

- shopping = ['apples', 'bread', 'milk']
- scores = [95, 87, 92, 78]
- mixed = ['Alice', 25, True, 3.14]

Lists can hold ANY type of data!

Creating Your First Lists

```
# Empty list (we'll add items later)
empty_list = []

# List of numbers
ages = [25, 32, 28, 45, 19]

# List of strings
names = ['Alice', 'Bob', 'Carol', 'David']

# List of mixed types (totally okay!)
person = ['Alice', 25, 'Engineer', True]

# Print them
print(ages)   # [25, 32, 28, 45, 19]
print(names)  # ['Alice', 'Bob', 'Carol', 'David']
```

Why Lists Are CRITICAL for ML

Machine Learning = processing lots of data

Lists in ML (you'll see this soon!):

- Training data: thousands of examples
- Features: multiple measurements per sample
- Predictions: results for many inputs
- Metrics: track accuracy over time

Path to ML:

- Python lists → NumPy arrays → Tensors

Master lists = foundation for everything!

This is THE most important data structure! ⭐

Lists vs Individual Variables

❌ Without lists (terrible!):

- score1 = 95
- score2 = 87
- score3 = 92
- ...
- score100 = 78 # This is insane!

✅ With lists (smart!):

- scores = [95, 87, 92, ..., 78]

Benefits:

- One variable for all data
- Easy to process
- Can loop through them
- Calculate statistics easily

List Indexing - Accessing Items

Like strings, lists have positions (indices)

Example: fruits = ['apple', 'banana', 'cherry']

- 'apple' 'banana' 'cherry'
- 0 1 2 ← Positions

Access items:

- fruits[0] → 'apple' (first)
- fruits[1] → 'banana' (second)
- fruits[2] → 'cherry' (third)

Remember: Python counts from 0!

First item is always position 0

Indexing Examples - Try These!

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

# Get first student
print(students[0])  # Alice

# Get third student
print(students[2])  # Carol

# Get last student (negative index!)
print(students[-1])  # Eve

# Get second to last
print(students[-2])  # David

# This works exactly like string indexing!
# Same rules apply
```

List Slicing - Getting Multiple Items

Slicing = extract a portion of the list

Format: list[start:stop]

Rules (same as strings!):

- start: included
- stop: NOT included
- Missing start: from beginning
- Missing stop: to end

This is ESSENTIAL for data processing:

- First 10 items
- Last 5 items
- Training set vs test set

List Slicing Examples

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# First 3 items
print(numbers[0:3])    # [0, 1, 2]
print(numbers[:3])     # [0, 1, 2] (shortcut)

# Last 3 items
print(numbers[-3:])    # [7, 8, 9]

# Middle items
print(numbers[3:7])    # [3, 4, 5, 6]

# Every other item
print(numbers[::2])    # [0, 2, 4, 6, 8]
# This is how you split data for ML!
# train = data[:800]  # First 80%
# test = data[800:]   # Last 20%
```

List Methods - Built-In Tools

Python has pre-made tools for lists:

- Adding items:
  - .append(item) - Add to end
  - .insert(position, item) - Add at position
- Removing items:
  - .remove(item) - Remove first occurrence
  - .pop() - Remove and return last item
- Organizing:
  - .sort() - Sort in place
  - .reverse() - Reverse order
- These modify the list directly!

append() - Adding Items to End

```
# Start with empty list
scores = []

# Add items one by one
scores.append(95)
print(scores)  # [95]
scores.append(87)
print(scores)  # [95, 87]
scores.append(92)
print(scores)  # [95, 87, 92]

# This is how you BUILD lists!
# Very common pattern in data processing
# You'll use append() constantly!
```

More List Methods in Action

```
tasks = ['email', 'meeting', 'lunch']

# Add to end
tasks.append('report')
print(tasks)  # ['email', 'meeting', 'lunch', 'report']

# Add at specific position
tasks.insert(1, 'coffee')
print(tasks)  # ['email', 'coffee', 'meeting', 'lunch', 'report']

# Remove specific item
tasks.remove('lunch')
print(tasks)  # ['email', 'coffee', 'meeting', 'report']

# Remove last item
last = tasks.pop()
print(last)   # 'report'
print(tasks)  # ['email', 'coffee', 'meeting']
```

Sorting and Reversing Lists

```
numbers = [42, 17, 93, 8, 55]

# Sort in ascending order (modifies list)
numbers.sort()
print(numbers)  # [8, 17, 42, 55, 93]

# Reverse the order
numbers.reverse()
print(numbers)  # [93, 55, 42, 17, 8]

# Sort names alphabetically
names = ['Charlie', 'Alice', 'Bob']
names.sort()
print(names)  # ['Alice', 'Bob', 'Charlie']
# Essential for data organization!
```

List Functions (Not Methods!)

Functions work ON lists (not part of them):

- len(list) - Count items
- sum(list) - Add up numbers
- min(list) - Find smallest
- max(list) - Find largest
- sorted(list) - Return sorted copy (doesn't change original)

Notice the difference:

- list.sort() - Changes list
- sorted(list) - Returns new sorted list

Both are useful!

List Functions for Data Analysis

```
scores = [95, 87, 92, 78, 88, 91]

# How many scores?
count = len(scores)
print(f'Number of scores: {count}')  # 6

# Total points
total = sum(scores)
print(f'Total points: {total}')  # 531

# Average score
average = total / count
print(f'Average: {average:.1f}')  # 88.5

# Highest and lowest
print(f'Highest: {max(scores)}')  # 95
print(f'Lowest: {min(scores)}')   # 78
# This is data analysis!
```

Lists Are Mutable (Can Change)

IMPORTANT difference from strings!

Strings are IMMUTABLE (can't change):

- name = 'Alice'
- name[0] = 'B' # ERROR!

Lists are MUTABLE (can change):

- names = ['Alice', 'Bob']
- names[0] = 'Carol' # Works!

This is powerful but be careful:

- Can modify items directly
- Changes affect the original
- Great for building data

Modifying List Items Directly

```
temperatures = [72, 68, 75, 70, 73]

# Change first temperature
temperatures[0] = 74
print(temperatures)  # [74, 68, 75, 70, 73]

# Change last temperature
temperatures[-1] = 71
print(temperatures)  # [74, 68, 75, 70, 71]

# Change multiple items with slicing
temperatures[1:3] = [69, 76]
print(temperatures)  # [74, 69, 76, 70, 71]
# This is data correction!
# Fix errors in your dataset
```

Real-World Example: Processing Data

Imagine you're analyzing customer ages:

Raw data: [25, 32, -5, 28, 150, 45, 19]

Problems:

- -5 is impossible (error)
- 150 is unlikely (outlier)

Your job:

- 1. Identify problems
- 2. Fix or remove them
- 3. Calculate statistics

This is real data cleaning!

- Companies pay for this skill!

Data Cleaning with Lists

```
# Raw data with errors
ages = [25, 32, -5, 28, 150, 45, 19]

# Remove negative age
ages.remove(-5)

# Remove outlier
ages.remove(150)

print(ages)  # [25, 32, 28, 45, 19]

# Now calculate statistics
average_age = sum(ages) / len(ages)
print(f'Average age: {average_age:.1f}')  # 29.8
# Clean data = accurate results!
# This is data science!
```

Part 2: Workshop

Build Your First Data Collections! • 25 minutes

💻 Exercise 1: Grade Manager (8 min)

Create file: grade_manager.py

Build a grade tracking system:

1. Create empty list: grades = []

2. Add these grades: 95, 87, 92, 78, 88

3. Calculate and print:

   - Number of grades
   - Highest grade
   - Lowest grade
   - Average grade

4. Sort grades from high to low

5. Print final sorted list

This is a real grade book!

Exercise 2: Shopping List Manager (9 min)

Create file: shopping_list.py

Build a smart shopping list:

1. Start with: items = ['apples', 'bread', 'milk']
2. Add 'eggs' to the list
3. Add 'cheese' to the list
4. Remove 'bread' (already have it)
5. Insert 'butter' at position 0 (need it first)
6. Print final list
7. Print how many items total
8. Print first and last items

Make it interactive and clear!

💻 Exercise 3: Temperature Analyzer

Create file: temp_analyzer.py

Analyze weekly temperatures:

temperatures = [72, 68, 75, 70, 73, 69, 71]

Calculate and print:

1. Average temperature
2. Highest temperature
3. Lowest temperature
4. Temperature range (high - low)
5. Days above average
6. Days below 70 degrees

Use f-strings for nice output!

This is data analysis!

Great Work!

You're managing data collections!

Part 3: GitHub - Share Your Code

Professional collaboration starts here!

What is GitHub? (Simple Explanation)

Git = Version control on YOUR computer

GitHub = Online platform for Git repositories

Think of GitHub as:

- Google Drive for code
- Facebook for programmers
- Portfolio for your projects

Why GitHub matters:

- Backup your code in the cloud
- Share projects with others
- Collaborate on teams
- Employers look at your GitHub!
- Most popular: 100+ million users!

Git vs GitHub - Clear Difference

Git:

- Software on your computer
- Tracks changes locally
- Works offline
- Created in 2005

GitHub:

- Website (github.com)
- Stores repositories online
- Requires internet
- Social features (follow, star, fork)
- Created in 2008

Analogy: Git = Word, GitHub = Google Docs

- You need both!

Creating Your GitHub Account

Step-by-step:

1. Go to github.com
2. Click 'Sign up'
3. Enter email (use professional email)
4. Create username (professional!)
   - Good: sarah-johnson, john-smith-dev
   - Bad: coolcoder123, iluvcats
5. Choose password
6. Verify email

FREE account is perfect!

This becomes your programming portfolio! 📂

Creating Your First Repository on GitHub

Repository = project folder on GitHub

Steps:

- 1. Log in to GitHub
- 2. Click green 'New' button (or + icon)
- 3. Enter repository name: python-fundamentals
- 4. Add description: 'My Python learning journey'
- 5. Choose Public (builds portfolio!)
- 6. DON'T initialize with README (we'll do it)
- 7. Click 'Create repository'

Done! You have an online repo! 🎉

Connecting Local Git to GitHub

```
# In your local project folder
# (Example: python_week2)

# 1. Add GitHub as 'remote' (online connection)
git remote add origin https://github.com/YOUR-USERNAME/python-fundamentals.git
# 'origin' is the standard name for main remote
# Replace YOUR-USERNAME with your GitHub username!

# 2. Verify it worked
git remote -v
# Should show:
# origin  https://github.com/YOUR-USERNAME/... (fetch)
# origin  https://github.com/YOUR-USERNAME/... (push)
```

git push - Upload to GitHub

push = upload your code to GitHub

Command: git push origin main

- origin: GitHub repository
- main: your branch name

First time push:

- git push -u origin main
- (-u sets default, only needed once)

After first time:

- git push
- (short version works!)

This uploads ALL your commits to GitHub!

Complete GitHub Workflow

```
# One-time setup (per project)
# 1. Create repo on GitHub
# 2. Connect local to GitHub
git remote add origin https://github.com/YOU/REPO.git

# 3. First push
git push -u origin main

# Daily workflow (after setup)
# 1. Write code
# 2. Save with Git
git add .
git commit -m 'Add feature X'

# 3. Upload to GitHub
git push
Your code is now online and backed up!
```

Why Push to GitHub? (Real Benefits)

1. Backup: Computer crashes? Code is safe online
2. Access anywhere: Work from any computer
3. Portfolio: Show employers your skills
4. Collaboration: Work with others later
5. Professional: Expected in tech jobs
6. Learning: See others' code, contribute

Employers WILL check your GitHub!

Build it from day 1! 💼

Your GitHub Profile = Your Portfolio

Your profile shows:

- All your public projects
- Contribution activity (green squares!)
- Programming languages you use
- How active you are

Tips for good profile:

- Commit regularly (green squares matter)
- Use clear project names
- Add README files (explain projects)
- Make important projects public

This is your programming resume! 

GitHub Best Practices for Beginners

1. Commit locally often (multiple times per day)
2. Push to GitHub daily (backup your work)
3. Write good commit messages (be professional)
4. Keep code organized (clean folders)
5. Add README files (explain what you built)

Avoid:

- ✗ Uploading passwords or secrets
- ✗ Huge files (images, datasets)
- ✗ Messy, uncommented code

Think: 'Would I be proud showing this to an employer?'

Part 4: Workshop

Push Your First Code to GitHub!

💻 Exercise 4: GitHub Setup & First Push

Get your code online!

Part A: Setup (10 min)

1. Create GitHub account (if not done)
2. Create repository: 'python-fundamentals'
3. Make it public
4. Copy the repository URL

Part B: Connect & Push (10 min)

1. In your local project folder:

   git remote add origin YOUR-URL
2. Check connection: git remote -v
3. Push your code: git push -u origin main
4. Visit github.com/YOUR-USERNAME/python-fundamentals
5. See your code online! 🎉

If errors: We'll help during workshop!

This is a big milestone!

Part 5: Quiz & Wrap-up

Session Complete!

Session 2.1 Quiz

🎉 Incredible Progress! Session 2.1 Complete

What you accomplished today:

- ✓ Learned lists (THE most important data structure)
- ✓ Created, indexed, and sliced lists
- ✓ Used list methods (append, sort, etc.)
- ✓ Calculated statistics on data
- ✓ Created GitHub account (portfolio!)
- ✓ Pushed code to GitHub (it's online!)

You're building real data analysis tools!

And you have an online portfolio! 🌟

This is MAJOR progress!

Homework - Practice Makes Perfect!

1. Complete the quiz (10 questions - required)
2. Project: Student Grade Analyzer
    - Build complete_grade_analyzer.py:
    - Start with empty list
    - Add 10 different grades
    - Calculate average, min, max
    - Count grades above 90 (A's)
    - Sort and display all grades
    - Use f-strings for professional output
3. Git & GitHub Practice
    - Commit your analyzer locally
    - Push to GitHub
    - View it online - take screenshot!
4. Prepare: Read about list comprehensions (next topic)

Next Session: List Comprehensions

Session 2.2: Advanced List Operations

You'll learn:

- List comprehensions (powerful shortcuts)
- Filtering data
- Transforming lists
- Nested lists (2D data)
- More data processing patterns

Why it matters:

- Faster code (fewer lines)
- Professional Python style
- Data transformation skills
- Foundation for NumPy

This takes your list skills to the next level! 🚀

You're Making Amazing Progress! 💪

Reflect on your journey:

Three sessions ago:

- 'What's a variable?'

Now:

- 'I write Python, use Git & GitHub, process data!'

You've learned:

- Variables, strings, lists
- Data processing and analysis
- Git for version control
- GitHub for online portfolio

This week: Data transformations, more patterns

Keep practicing! You're doing great! 🎓

See you next session!