CE49X: Introduction to Computational Thinking and Data Science for Civil Engineers

Week 3: Python Modules, Strings, and Data Science Tools

Dr. Eyuphan Koc

Department of Civil Engineering
Bogazici University

Fall 2025

Based on "A Whirlwind Tour of Python" by Jake VanderPlas
Chapters 13-15: Modules & Packages, Strings & Regular Expressions, Data Science Tools Preview
https://github.com/jakevdp/WhirlwindTourOfPython

Week 3 Outline

Python's "Batteries Included" Philosophy

What Makes Python Powerful?

- **Built-in modules**: Ready-to-use functionality
- **Third-party packages**: Extensive ecosystem (100,000+ packages)
- **Easy installation**: Package managers like pip and conda
- **Modular design**: Organize code into reusable components

Civil Engineering Applications

- Access FEM analysis libraries (OpenSeesPy, FEniCS)
- Use structural design modules (PyNite, StructPy)
- Integrate with BIM software (IfcOpenShell)
- Build custom analysis tools for your specific needs

Today's Goal

Master Python's module system to leverage existing tools and build your own!

Understanding Modules vs Packages

Module

- Single Python file (.py)
- Contains functions, classes, variables
- Example: `beam_analysis.py`
- Imported with `import beam_analysis`

```python
# beam_analysis.py
def calculate_moment(load, span):
    return load * span**2 / 8
```

Package

- Directory containing multiple modules
- Has `__init__.py` file
- Example: `structural/` package
- Contains many sub-modules

```python
# structural/
#   __init__.py
#   beams.py
#   columns.py
#   loads.py
```

Think of it this way

Module = One tool (hammer) | Package = Toolbox (complete set)

1. Explicit Module Import (Recommended)

Preserves namespace clarity, easy to trace function origins, prevents naming conflicts.

```
import math
import statistics  # <-- Try importing this!

# Clear where functions come from
beam_angle = math.cos(math.pi/4)  # From math module
load_avg = statistics.mean([10, 15, 20, 25])  # From statistics

print(f"Angle: {beam_angle:.3f}")
print(f"Average load: {load_avg} kN")
```

[QUICK] Try it yourself (1 minute)

Import the `datetime` module and print today's date! Hint: Use `datetime.date.today()`

2. Module Import with Alias

Shorter names, standard scientific conventions, maintains namespace separation.

```python
import numpy as np  # Standard alias  # --> Predict: What will np stand for?
import matplotlib.pyplot as plt  # For plotting
import pandas as pd  # For data analysis

# Standard scientific Python conventions
loads = np.array([120, 145, 98, 167, 134])  # kN
stresses = loads / 25  # MPa (assuming area = 25 cm^2)

print(f"Max stress: {np.max(stresses):.1f} MPa")
print(f"Mean stress: {np.mean(stresses):.1f} MPa")
```

Community Conventions

- `np` = NumPy (always)
- `pd` = Pandas (always)
- `plt` = Matplotlib.pyplot (always)
- Following these makes your code instantly recognizable!

Selective Imports

3. Import Specific Functions

Import only what you need, cleaner code, good for frequently used functions.

```python
from math import cos, sin, pi, sqrt
from statistics import mean, stdev

# Structural vibration calculation  # <-- Students: Can you spot the issue?
frequency = 2.5  # Hz
period = 1 / frequency
angle = 2 * pi * frequency * 0.1  # at t=0.1s

# Direct use without module prefix
amplitude = 10 * cos(angle)  # mm
velocity = -10 * 2 * pi * frequency * sin(angle)  # mm/s

print(f"Displacement: {amplitude:.2f} mm")
print(f"Velocity: {velocity:.2f} mm/s")
```

[DEBUG] Common Pitfall

What happens if you do: `from numpy import *` and then `sum([1, 2, 3])`? The NumPy sum overwrites Python's built-in sum!

[LIVE] Coding Challenge 2: Module Organization

Your Task (4 minutes)

Create a structural analysis module organization:

- Define functions for beam moment, shear, and deflection
- Import and use them correctly
- Calculate values for a 6m beam with 10 kN/m load

Starter Code

```python
# Define your analysis functions
def calculate_moment(w, L):
    # YOUR CODE HERE: return max moment for simply supported beam
    pass

def calculate_shear(w, L):
    # YOUR CODE HERE: return max shear
    pass

def calculate_deflection(w, L, E=200000, I=8333):
    # YOUR CODE HERE: return max deflection
    pass

# Test with: w=10 kN/m, L=6m
```

Import Wildcards: When to Avoid

4. Wildcard Imports (Generally Avoid)

`from module import *` imports everything, can overwrite functions, makes debugging harder.

The Problem - A Real Debugging Nightmare

- Python's `sum([1, 2, 3])` returns 6
- NumPy's `sum([1, 2, 3])` returns `numpy.int64(6)`
- Different behavior can break your code silently!

Best Practice

- **Good**: `import numpy as np` or `from math import sin, cos`
- **Bad**: `from numpy import *`
- Exception: Interactive exploration in Jupyter (but fix before production)

Essential Standard Library Modules for Engineers

Mathematical & Scientific

- `math`: Trigonometry, logarithms
- `statistics`: Mean, stdev, regression
- `random`: Monte Carlo simulations
- `cmath`: Complex numbers (signals)

Data Processing

- `csv`: Read sensor data files
- `json`: API data exchange
- `pickle`: Save Python objects

System & File Operations

- `os`: File system navigation
- `pathlib`: Modern path handling
- `datetime`: Time series data

Advanced Tools

- `itertools`: Combinations, permutations
- `functools`: Caching, decorators
- `urllib`: Download data from web

[PRACTICE] Try This Now (2 minutes)

Import `random` and generate 5 random concrete strengths between 20-40 MPa

[TOGETHER] Installing Third-Party Packages

Package Installation Methods

- **pip**: `pip install numpy scipy matplotlib`
- **conda**: `conda install numpy scipy matplotlib`
- **Virtual environments**: Isolated project dependencies

```
# Check installed packages
import sys
!pip list | grep numpy  # In Jupyter/Colab
# Install specific version: pip install numpy==1.21.0
# Install from requirements: pip install -r requirements.txt
```

Engineering Package Examples

- `pip install openseespy` – Structural analysis
- `pip install pynite` – Frame analysis
- `pip install ifcopenshell` – BIM/IFC files

Professional Tip

Create a `requirements.txt` file for each project to track dependencies!

Why String Processing Matters in Engineering

Common Engineering String Tasks

- **Data Import**: Reading CSV files, parsing measurement data
- **Report Generation**: Creating formatted output documents
- **File Processing**: Handling CAD files, analysis output files
- **Data Validation**: Checking input formats, units, ranges
- **Database Operations**: SQL queries, data cleaning

Real-World Examples

- Parsing bridge inspection reports from text files
- Extracting coordinates from survey data
- Formatting structural analysis results for presentations
- Processing weather station data files
- Converting between different data formats

Python String Fundamentals

String Definition Options

- **Single quotes**: `'Steel Grade 250'`
- **Double quotes**: `"Concrete fc'=30"` (use when string contains ')
- **Triple quotes**: Multi-line strings, docstrings
- **Raw strings**: `r"C:\Bridge\Data\sensors.csv"` (no escapes)

```python
# Engineering data with different string types
material = 'reinforced concrete'  # Simple string
spec = "fc'=30 MPa"  # Contains apostrophe  # <-- Why double quotes?
report = """Structural Analysis Report
Date: 2025-01-15
Project: Highway Bridge"""

# File paths - always use raw strings on Windows!
data_file = r"C:\Projects\Bridge_2025\load_data.csv"  # Correct
# bad_path = "C:\Projects\Bridge_2025\load_data.csv"  # Will fail!
```

[DEBUG] Common Error

What's wrong with: `path = "C:\newfolder\test.txt"`? The \n becomes newline! Use raw strings: `r"C:\newfolder\test.txt"`

Essential String Methods for Engineering Data

```python
# Processing messy engineering data  # --> Predict the outputs!
material = "  REINFORCED conCRETE  "
clean = material.strip().title()
print(clean)  # ?

# Parsing member IDs
member_id = "BEAM-B01-LEVEL3"
parts = member_id.split('-')  # ['BEAM', 'B01', 'LEVEL3']
beam_type = parts[0].lower()  # 'beam'
beam_num = parts[1]  # 'B01'

# Replacing units
measurement = "Load: 1500 kips"
metric = measurement.replace('kips', 'kN')
value = float(measurement.split()[1]) * 4.448  # Convert to kN
print(f"Metric: {value:.1f} kN")
```

Key String Methods

`.strip()` removes whitespace | `.split()` creates list | `.join()` combines list | `.replace()` substitutes text

[PRACTICE] String Cleaning Challenge

Your Task (3 minutes)

Clean and standardize this sensor data:

Starter Code

```python
# Messy sensor readings
data = [
    "  Sensor_01: 125.3 mPa  ",
    "sensor_02:98.7mpa",
    "SENSOR_03 : 145.2 MPA",
    "  sensor-04: 112.8 Mpa"
]

# YOUR TASK: Clean and standardize to format "S01: 125.3 MPa"
# Hints:
# 1. Extract sensor number
# 2. Extract value
# 3. Standardize units to "MPa"
# 4. Format as "S##: value MPa"

for reading in data:
    # YOUR CODE HERE
    pass
```

Modern String Formatting: f-strings

Why f-strings? (Python 3.6+)

Readable, fast, supports expressions, type formatting

```python
# Engineering calculations with formatted output
beam_id = "B-001"
moment = 245.678  # kN*m
capacity = 300  # kN*m
utilization = moment / capacity * 100

# f-string with expressions  # <-- Try changing precision!
report = f"""Beam Analysis Report
ID: {beam_id}
Moment: {moment:.1f} kN*m
Capacity: {capacity} kN*m
Utilization: {utilization:.1f}%
Status: {'OK' if utilization < 90 else 'CHECK REQUIRED'}"""

print(report)

# Advanced formatting
pi_value = 3.14159265
print(f"Pi to 3 decimals: {pi_value:.3f}")
print(f"Percentage: {0.856:.1%}")  # Automatic % conversion!
print(f"Scientific: {1234567:.2e}")
```

Regular Expressions

What are Regular Expressions?

Pattern matching for complex text processing: find patterns, validate formats, extract data.

Engineering Applications

- Extract dimensions: "200x400x6000mm"
- Validate formats: coordinates, member IDs, loads
- Parse log files and reports
- Clean measurement data

Learning Tip

Start with simple patterns, build complexity gradually.

Pattern Matching Power

Extract complex patterns from text: measurements, IDs, coordinates

```python
import re

# Engineering report with mixed data
report = """Bridge inspection 2025-01-15:
Beam B-001: 250x400mm, stress 145.3 MPa
Column C-42A: 600x600mm, load 1250.5 kN
Coordinates: (42.3567, -71.0589)
Next inspection: 2025-07-15"""

# Extract all measurements with units  # --> Try these patterns!
nums_with_units = re.findall(r'\d+\.?\d*\s*(?:mm|MPa|kN)', report)
print("Measurements:", nums_with_units)

# Extract member IDs (letter-numbers-optional letter)
member_ids = re.findall(r'[A-Z]-\d+[A-Z]?', report)
print("Members:", member_ids)

# Extract dates (YYYY-MM-DD)
dates = re.findall(r'\d{4}-\d{2}-\d{2}', report)
print("Dates:", dates)
```

[DEBUG] Together: Fix the Regex Patterns

Collaborative Debugging (4 minutes)

These patterns have bugs. Let's fix them together!

```python
import re

# Bug #1: Coordinate extraction
coords = "Location: (42.3567, -71.0589) and (40.7128, -74.006)"
pattern1 = r'\d+\.\d+, \d+\.\d+'  # What's missing?  # <-- Students: Fix this!

# Bug #2: Dimension extraction
dims = "Steel section: 200x300x6000mm, Concrete: 400x800mm"
pattern2 = r'\d+x\d+'  # Doesn't get all dimensions

# Bug #3: Load value extraction
loads = "Dead: 125.5kN, Live: 85 kN, Wind: 42.75kN"
pattern3 = r'\d+kN'  # Missing something?

# Test and fix each pattern
print("Coords found:", re.findall(pattern1, coords))
print("Dimensions found:", re.findall(pattern2, dims))
print("Loads found:", re.findall(pattern3, loads))
```

Discuss: What patterns did you find? How would you fix them?

Mini-Competition (5 minutes)

Extract all required data from this construction log. Most complete & efficient wins!

```python
log = """2025-01-15 08:30:45 - Pour started: Section A1-B2
Concrete: fc'=35MPa, Slump: 150mm, Temp: 22.5C
Volume: 45.5m3, Truck#: CT-2847, Driver: ID#8934
2025-01-15 11:45:20 - Pour complete: Section A1-B2
Total time: 3.25hrs, Weather: Clear, 18.5C
Next pour: Section B2-C3, Date: 2025-01-16"""

# YOUR CHALLENGE: Extract:
# 1. All timestamps (YYYY-MM-DD HH:MM:SS)
# 2. All section IDs (A1-B2 format)
# 3. All measurements with units
# 4. All ID numbers (ID#XXXX or CT-XXXX)

# YOUR CODE HERE - Aim for elegance!


# Scoring: Correctness (50%), Code efficiency (30%), Readability (20%)
```

The Scientific Python Ecosystem

OpenSeesPy FEniCS PyNite
Pandas Scikit-learn SymPy Jupyter
NumPy SciPy Matplotlib
Python Core Language

Build on Giants' Shoulders

Don't reinvent the wheel - use tested, optimized libraries!

NumPy: Foundation of Scientific Python

Key Features

N-dimensional arrays, vectorized operations, linear algebra, 10-100x faster than Python lists.

Engineering Applications

- Store sensor measurement data
- Represent structural matrices
- Perform finite element calculations
- Process time-series monitoring data

[LIVE] NumPy: Foundation of Scientific Computing

Why NumPy?

10-100x faster than Python lists, foundation for all scientific libraries

```python
import numpy as np

# Structural loads analysis  # --> Predict: What's the speedup?
loads_list = [120, 145, 98, 167, 134]  # Python list
loads_array = np.array(loads_list)     # NumPy array

# Compare operations
# Python way (slow)
factored_list = [x * 1.5 for x in loads_list]

# NumPy way (fast) - vectorized!
factored_array = loads_array * 1.5  # All at once!

print(f"Max load: {np.max(loads_array)} kN")
print(f"Mean: {np.mean(loads_array):.1f} kN")
print(f"Std Dev: {np.std(loads_array):.1f} kN")

# Matrix operations for structural analysis
K = np.array([[1000, -500], [-500, 1000]])  # Stiffness matrix
F = np.array([100, 50])  # Force vector
u = np.linalg.solve(K, F)  # Solve K*u = F
print(f"Displacements: {u}")
```

[PRACTICE] NumPy Challenge: Beam Analysis

Your Task (4 minutes)

Use NumPy to analyze a simply supported beam with multiple point loads

Starter Code

```python
import numpy as np

# Beam data
L = 10  # meters
loads = np.array([50, 30, 40, 20])  # kN
positions = np.array([2, 4, 6, 8])  # meters from left support

# YOUR TASK:
# 1. Calculate reactions at supports (RA and RB)
# 2. Calculate moment at each load position
# 3. Find maximum moment and its location

# Hint: For simply supported beam:
# RA = sum(P * (L - x)) / L
# M(x) = RA * x - sum(P) for loads before x

# YOUR CODE HERE
```

Pandas: Data Analysis

Key Features

DataFrame structures, data cleaning, import/export (CSV, Excel), grouping, time series.

Engineering Applications

- Import sensor data and test results
- Analyze time series monitoring
- Create statistical summaries
- Generate analysis reports

Why Pandas?

Excel-like functionality with Python power and reproducibility.

[TOGETHER] Pandas: Data Analysis Made Easy

```python
import pandas as pd
import numpy as np

# Create concrete test data  # <-- Students: Follow along!
concrete_data = {
    'sample_id': ['C001', 'C002', 'C003', 'C004', 'C005'],
    'age_days': [7, 7, 28, 28, 28],
    'strength_mpa': [18.5, 19.2, 28.3, 31.5, 29.8],
    'mix_type': ['A', 'A', 'B', 'B', 'B']
}

df = pd.DataFrame(concrete_data)
print(df)

# Analysis operations
print(f"\nMean 28-day strength: {df[df['age_days']==28]['strength_mpa'].mean():.1f} MPa")

# Group by mix type
stats = df.groupby('mix_type')['strength_mpa'].agg(['mean', 'std', 'count'])
print("\nStatistics by mix:")
print(stats)

# Add compliance check
df['compliant'] = df['strength_mpa'] >= 25
print(f"\nCompliance rate: {df['compliant'].mean():.1%}")
```

Key Features

Publication-quality plots, wide variety of chart types, complete customization, multiple output formats.

Engineering Visualizations

- Load-displacement curves
- Time series sensor data
- Stress distribution plots
- Material property distributions
- Project progress charts

[LIVE] Matplotlib: Publication-Quality Plots

```python
import matplotlib.pyplot as plt
import numpy as np

# Generate structural response data  # --> Try different parameters!
time = np.linspace(0, 10, 100)  # seconds
frequency = 2.0  # Hz
damping = 0.1
amplitude = 10 * np.exp(-damping * time) * np.sin(2 * np.pi * frequency * time)

# Create professional plot
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

# Time series
ax1.plot(time, amplitude, 'b-', linewidth=2)
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Displacement (mm)')
ax1.set_title('Damped Vibration Response')
ax1.grid(True, alpha=0.3)
ax1.axhline(y=0, color='k', linestyle='-', linewidth=0.5)

# Histogram of peaks
peaks = amplitude[amplitude > 0]
ax2.hist(peaks, bins=20, edgecolor='black', alpha=0.7)
ax2.set_xlabel('Amplitude (mm)')
ax2.set_ylabel('Frequency')
ax2.set_title('Distribution of Positive Peaks')

plt.tight_layout()
plt.show()
```

[QUICK] SciPy: Advanced Scientific Computing

Key Capabilities

Optimization, curve fitting, interpolation, integration, signal processing

```python
from scipy import optimize, interpolate
import numpy as np

# Curve fitting example - concrete strength vs time
days = np.array([3, 7, 14, 28, 56])
strength = np.array([12, 20, 26, 30, 32])  # MPa

# Fit exponential model: f(t) = a * (1 - exp(-b*t))
def model(t, a, b):
    return a * (1 - np.exp(-b * t))

params, _ = optimize.curve_fit(model, days, strength)
print(f"Model: f(t) = {params[0]:.1f} * (1 - exp(-{params[1]:.3f}*t))")

# Interpolation for intermediate values
interp_func = interpolate.interp1d(days, strength, kind='cubic')
day_21_strength = interp_func(21)
print(f"Predicted 21-day strength: {day_21_strength:.1f} MPa")
```

Pair Exercise (2 minutes)

With your neighbor: What other SciPy functions would be useful for structural analysis?

[EXPLORE] Specialized Engineering Packages

Structural Analysis

- `OpenSeesPy`: Earthquake engineering
- `PyNite`: 3D frame analysis
- `anaStruct`: 2D frame analysis
- `FEniCS`: Finite element modeling

Data & Visualization

- `Plotly`: Interactive plots
- `Seaborn`: Statistical graphics
- `Bokeh`: Web visualizations

BIM & CAD

- `IfcOpenShell`: IFC file handling
- `PythonOCC`: CAD kernel
- `FreeCAD API`: CAD automation

Utilities

- `Pint`: Unit conversions
- `SymPy`: Symbolic math
- `Uncertainties`: Error propagation

[DISCUSS] Class Poll

Which package sounds most useful for your projects? Why?

[TOGETHER] Getting Started with Data Science

Installation Options

- **Anaconda**: Complete scientific distribution (recommended)
- **Miniconda**: Minimal conda installer
- **pip + venv**: Traditional Python approach

```python
# Check your setup
import sys
print(f"Python: {sys.version}")
# Check essential packages
try:
    import numpy as np
    import pandas as pd
    import matplotlib
    print("Essential packages installed!")
    print(f"NumPy version: {np.__version__}")
except ImportError as e:
    print(f"Missing package: {e}")
```

Professional Workflow

1. Create virtual environment | 2. Install packages | 3. Save requirements.txt | 4. Version control

Real Engineering Workflow: From Data to Decision

Complete Data Science Pipeline

1. **Import**: Multiple data sources (CSV, Excel, databases)
2. **Clean**: Handle missing values, outliers, units
3. **Transform**: Calculate derived properties
4. **Analyze**: Statistical analysis, pattern detection
5. **Visualize**: Create publication-quality figures
6. **Report**: Generate automated reports

Today's Case Study

Bridge monitoring data: 1000+ sensors, 6 months of data, multiple formats

- Challenge: Identify anomalies and predict maintenance needs
- Tools: Pandas for processing, NumPy for calculations, Matplotlib for visualization

Industry Reality

80% of data science is data cleaning - let's automate it!

[COMPETITION] Data Pipeline Challenge

Team Competition (7 minutes)

Build the most efficient concrete quality control pipeline!

```python
import pandas as pd
import numpy as np

# Sample data (you'll get more)
data = {
    'batch_id': ['B001', 'B002', 'B003', 'B004', 'B005'],
    'strength_7d': [18.5, 'N/A', 21.3, 19.8, 22.1],
    'strength_28d': [28.3, 31.5, '29.8', 27.2, 33.4],
    'cement_kg': [320, 350, 340, 330, 360],
    'water_cement': [0.45, 0.42, 0.43, 0.44, 0.40]
}

# YOUR CHALLENGE:
# 1. Clean the data (handle 'N/A', convert types)
# 2. Calculate strength gain ratio (28d/7d)
# 3. Flag non-compliant batches (28d < 30 MPa)
# 4. Find correlation between w/c ratio and strength
# 5. Create summary statistics by cement content group

# YOUR CODE HERE - Most complete & elegant wins!
```

[LIVE] Complete Analysis Pipeline

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Real-world pipeline  # --> Follow along and modify!
def analyze_concrete_data(filename):
    # Import
    df = pd.read_csv(filename)

    # Clean
    df['strength'] = pd.to_numeric(df['strength'], errors='coerce')
    df = df.dropna()

    # Transform
    df['strength_ratio'] = df['strength'] / df['target_strength']
    df['compliance'] = df['strength_ratio'] >= 1.0

    # Analyze
    summary = {
        'total_samples': len(df),
        'mean_strength': df['strength'].mean(),
        'std_strength': df['strength'].std(),
        'compliance_rate': df['compliance'].mean(),
        'critical_samples': df[~df['compliance']]['sample_id'].tolist()
    }

    return df, summary

# Test the pipeline
# df, results = analyze_concrete_data('test_data.csv')
```

What We've Covered This Week

Modules & Packages

- Import strategies and best practices
- Standard library modules
- Package installation with pip/conda
- Creating modular code

String Processing

- String methods for data cleaning
- Modern formatting with f-strings
- Regular expressions for pattern matching
- Text extraction from engineering data

Data Science Foundations

- NumPy for numerical computing
- Pandas for data analysis
- Matplotlib for visualization
- SciPy for scientific computing

Practical Applications

- Complete data pipelines
- Engineering data processing
- Quality control automation
- Report generation

Key Programming Concepts Mastered

Core Python Skills

- **Modular Programming**: Organizing code with modules and packages
- **Text Processing**: String manipulation and regex for data extraction
- **Scientific Computing**: NumPy arrays and vectorized operations
- **Data Analysis**: Pandas DataFrames for structured data
- **Visualization**: Creating publication-quality plots

Engineering Problem-Solving Skills

- Process sensor data from multiple sources
- Clean and standardize messy engineering data
- Perform statistical analysis on test results
- Create automated quality control pipelines
- Generate professional technical reports

Questions?

Thank you!

Dr. Eyuphan Koc
eyuphan.koc@bogazici.edu.tr