Git & Python Lab Session

Course: DATA1300
Week: 4

Lab Overview

In this lab, you will practice **professional Git workflows** by creating a Python project with proper directory structure, making changes on a feature branch, and merging back to main. This mirrors real-world development practices used at companies worldwide.

By the end of this lab, you will:

- Create a Python project with professional src/ directory structure
- Initialize a Git repository and push to GitHub
- Create and work on feature branches
- Commit changes with meaningful messages
- Merge branches back to main
- Delete branches after merging
- Understand why professional structure matters

Part 1: Project Setup

Step 1.1: Create Project Directory Structure

Open your terminal/Command Prompt and execute these commands:

Create project folder

mkdir temperature-converter
cd temperature-converter

Create professional directory structure

mkdir -p src/temp_app tests data
touch src/__init__.py

touch src/temp_app/__init__.py
touch README.md
touch requirements.txt
touch .gitignore

What you just created:

```
temperature-converter/
├── src/ # Source code directory
│   ├── __init__.py # Makes src a package
│   └── temp_app/ # Main application package
│   └── __init__.py # Makes temp_app a package
├── tests/ # Tests directory (empty for now)
├── data/ # Data files directory (empty for now)
├── README.md  # Project documentation
├── requirements.txt # Python dependencies list
└── .gitignore # Files Git should ignore
```

Why this structure?

- **src/** folder = Isolates your code from configuration files
- **__init__.py** files = Makes directories into Python packages (required for imports)
- **tests/** = Future home for automated tests
- **data/** = Where we'll store CSV files later
- **.gitignore** = Prevents committing unnecessary files (venv/, **pycache**)

Step 1.2: Create Your First Python File

Create the main Python program that will convert temperatures:

File: src/temp_app/converter.py

"""
Temperature Converter Module
Provides functions to convert between Celsius, Fahrenheit, and Kelvin.
"""

def celsius_to_fahrenheit(celsius):
"""
Convert temperature from Celsius to Fahrenheit.

```
Args:
    celsius (float): Temperature in Celsius

Returns:
    float: Temperature in Fahrenheit

Example:
    >>> celsius_to_fahrenheit(0)
```

```
    32.0
    >>> celsius_to_fahrenheit(100)
    212.0
"""
return (celsius * 9/5) + 32
```

def fahrenheit_to_celsius(fahrenheit):
"""
Convert temperature from Fahrenheit to Celsius.

```
Args:
    fahrenheit (float): Temperature in Fahrenheit

Returns:
    float: Temperature in Celsius

Example:
    >>> fahrenheit_to_celsius(32)
    0.0
    >>> fahrenheit_to_celsius(212)
    100.0
"""
return (fahrenheit - 32) * 5/9
```

def celsius_to_kelvin(celsius):
"""
Convert temperature from Celsius to Kelvin.

```
Args:
    celsius (float): Temperature in Celsius

Returns:
    float: Temperature in Kelvin

Example:
    >>> celsius_to_kelvin(0)
    273.15
"""
return celsius + 273.15
```

def kelvin_to_celsius(kelvin):
"""
Convert temperature from Kelvin to Celsius.

```
Args:
    kelvin (float): Temperature in Kelvin

Returns:
    float: Temperature in Celsius
```

```
Example:
    >>> kelvin_to_celsius(273.15)
    0.0
"""
return kelvin - 273.15
```

if **name** == "**main**":
# Quick test of conversion functions
print("Temperature Converter Tests")
print(f"0°C = {celsius_to_fahrenheit(0)}°F")
print(f"100°C = {celsius_to_fahrenheit(100)}°F")
print(f"0°C = {celsius_to_kelvin(0)}K")

Step 1.3: Create README.md

Document your project:

File: README.md

Temperature Converter

A simple Python application to convert temperatures between Celsius, Fahrenheit, and Kelvin.

Installation

1. Clone this repository:
   git clone <your-repo-url>
   cd temperature-converter

2. Create a virtual environment:
   python3 -m venv venv
   source venv/bin/activate # macOS/Linux
   venv\Scripts\activate # Windows

3. Install dependencies (if any):
   pip install -r requirements.txt

Usage

python src/temp_app/converter.py

Features

- Convert Celsius ↔️ Fahrenheit
- Convert Celsius ↔️ Kelvin
- Clean, documented code

- Professional project structure

Author

Your Name

Step 1.4: Create .gitignore

Tell Git which files to ignore:

Edit the file and add the following.

Virtual environment

venv/
env/
ENV/

Python bytecode

pycache/
*.pyc
*.pyo
*.pyd
.Python

IDE

.vscode/
.idea/
*.swp
*.swo

OS

.DS_Store
Thumbs.db

Project

.env
config.local
*.log

Step 1.5: Create requirements.txt

For now, this will be empty (no external dependencies yet):

Step 1.6: Initialize Git

Initialize Git repository

git init

Check status

git status

Add all files

git add .

Create first commit

git commit -m "Initial commit: Project structure with temperature converter functions"

Verify commit

git log --oneline

Expected output:

- <commit id> Initial commit: Project structure with temperature converter functions

---

Step 1.7: Test Your Code

Before pushing, verify your Python code runs:

Test from the project root directory

python src/temp_app/converter.py

Expected output:
Temperature Converter Tests
0°C = 32.0°F
100°C = 212.0°F
0°C = 273.15K

If this works, you're ready for part 2

Part 2: Branching, Committing & Merging

Step 2.1: Create a Feature Branch

Imagine you want to add a new feature: conversion to Rankine scale. Professional developers create a separate branch for new features:

Create and switch to new branch

git checkout -b feature/add-rankine-conversion

Verify you're on the new branch

git branch

Output should show:

main

* feature/add-rankine-conversion

What's happening:

- `git checkout -b feature/...` = Create new branch AND switch to it
- The `-b` flag means "create if doesn't exist"
- Your main branch stays clean; all changes happen on feature branch

---

Step 2.2: Update Your Code on the Feature Branch

Add a new conversion function to your converter.py file.

Edit **src/temp_app/converter.py** and add this function after the kelvin_to_celsius function:

Add this code at the end of the function definitions (before `if __name__ == "__main__":`

def celsius_to_rankine(celsius):
"""
Convert temperature from Celsius to Rankine.

```
Formula: Rankine = (Celsius + 273.15) × 9/5

Args:
    celsius (float): Temperature in Celsius

Returns:
    float: Temperature in Rankine

Example:
    >>> celsius_to_rankine(0)
    491.67
"""
kelvin = celsius_to_kelvin(celsius)
return kelvin * 9/5
```

def rankine_to_celsius(rankine):
"""
Convert temperature from Rankine to Celsius.

```
Formula: Celsius = (Rankine × 5/9) - 273.15

Args:
    rankine (float): Temperature in Rankine
```

```
Returns:
    float: Temperature in Celsius

Example:
    >>> rankine_to_celsius(491.67)
    0.0
"""
kelvin = rankine * 5/9
return kelvin_to_celsius(kelvin)
```

Update the test section at the bottom:

Replace:
if **name** == "**main**":
# Quick test of conversion functions
print("Temperature Converter Tests")
print(f"0°C = {celsius_to_fahrenheit(0)}°F")
print(f"100°C = {celsius_to_fahrenheit(100)}°F")
print(f"0°C = {celsius_to_kelvin(0)}K")

With:
if **name** == "**main**":
# Quick test of conversion functions
print("Temperature Converter Tests")
print(f"0°C = {celsius_to_fahrenheit(0)}°F")
print(f"100°C = {celsius_to_fahrenheit(100)}°F")
print(f"0°C = {celsius_to_kelvin(0)}K")
print(f"0°C = {celsius_to_rankine(0)}°R")
print(f"491.67°R = {rankine_to_celsius(491.67)}°C")

Step 2.3: Test the New Feature

Run the updated code

python src/temp_app/converter.py

Expected output:
Temperature Converter Tests
0°C = 32.0°F
100°C = 212.0°F
0°C = 273.15K
0°C = 491.67°R
491.67°R = 0.0°C

Step 2.4: Stage, Commit, and Push

Stage your changes:

git status

Shows: modified: src/temp_app/converter.py

git add src/temp_app/converter.py

Commit with a descriptive message:

git commit -m "Add Rankine temperature conversion functions

- Implement celsius_to_rankine() conversion
- Implement rankine_to_celsius() conversion
- Update test cases with Rankine examples
- All conversions tested and verified"

Why detailed commit messages?

- First line = Quick summary (50 chars)
- Blank line = Separator
- Additional lines = What and why
- Other developers can understand your changes

Verify the commit:

git log --oneline -3

Output shows your new commit on feature/add-rankine-conversion

---

Step 2.5: Create a Pull Request (Review Simulation)

In real teams, Pull Requests (PRs) allow code review. For this lab, we'll simulate the process locally.

First, push your feature branch to GitHub:

git push -u origin feature/add-rankine-conversion

Note: This assumes you've already connected to GitHub

If not, go to Step 1.7 in Part 1 Lab Guide first

Check GitHub website:

- Visit your repository
- GitHub should show a "Compare & Pull Request" button
- Click it and add a description:

Title: Add Rankine temperature conversion

Description:
Adds conversion functions for Rankine scale, commonly used in engineering.

Changes:

- New functions: celsius_to_rankine(), rankine_to_celsius()
- Updated test cases to verify Rankine conversions
- All tests passing

Ready to merge.

Step 2.6: Merge the Feature Branch to Main

Switch back to main branch:

git checkout main

Verify you're on main:

git branch

Output:

* main

feature/add-rankine-conversion

Merge the feature branch:

git merge feature/add-rankine-conversion

Output should show:

Merge made by the 'recursive' strategy.

src/temp_app/converter.py | 42 insertions(+)

1 file changed, 42 insertions(+)

Verify the merge:

git log --oneline -5

Should show both your feature commits and merge commit

Test that main branch has the new code:

python src/temp_app/converter.py

Should output all conversions including Rankine

Step 2.7: Delete the Feature Branch

After merging, clean up by deleting the feature branch:

Delete locally:

git branch -d feature/add-rankine-conversion

Verify it's deleted

git branch

Should only show: * main

Delete from GitHub (if you pushed it):

git push origin --delete feature/add-rankine-conversion

Why delete branches?

- Keeps repository clean
- Prevents confusion with old branches
- Industry standard practice
- Branch is merged, so no work is lost

Step 2.8: Push to GitHub

Finalize by pushing your merged main branch to GitHub:

git push origin main

Verify on GitHub website

Go to your repository on Github

Should see:

- Rankine functions in converter.py

- Updated README

- Professional directory structure

Quick Command Reference

Branching

git branch # List all branches
git branch -a # List local + remote branches
git checkout -b feature/name # Create and switch to new branch
git checkout main # Switch to main branch
git branch -d feature/name # Delete branch (safe)
git branch -D feature/name # Force delete branch (unsafe)

Committing

git status # See what changed
git add <file> # Stage specific file
git add . # Stage all changes
git commit -m "message" # Commit with message
git log --oneline # See commit history (compact)
git log --oneline -5 # See last 5 commits

Merging

git merge feature/name # Merge branch into current branch

Pushing

git push -u origin feature/name # Push branch first time (with tracking)
git push origin feature/name # Push branch (after first push)
git push origin main # Push main branch
git push origin --delete feature/name # Delete remote branch

Common Issues & Solutions

Issue: "fatal: not a git repository"

Solution: Make sure you're in the right directory:
cd temperature-converter
git status # Should work now

Issue: "error: Your local changes to the following files would be overwritten by checkout"

Solution: Commit your changes first:
git add .
git commit -m "Save work in progress"
git checkout main

Issue: "error: pathspec 'feature/add-rankine-conversion' did not match any file(s)"

Solution: The branch doesn't exist or you misspelled it:
git branch # List actual branch names
git checkout -b feature/add-rankine-conversion # Create it

Issue: Can't merge - "CONFLICT" message

Solution: Git found conflicting changes. Edit the conflicting file, remove conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`), and commit:

Edit file to fix conflicts

git add .
git commit -m "Resolve merge conflict"
